Making Everything Easier!

MarkLogic Special Edition

Semantics

FOR DUMMIES®

A Wiley Brand

Learn:

- **Why semantic technology is gaining traction now**
- **How semantics is solving major challenges in the real world**
- **How to model data as RDF and query with SPARQL**

Brought to you by

MarkLogic®

Allen Taylor

Semantics

FOR DUMMIES®

A Wiley Brand

MarkLogic Special Edition

by Allen Taylor

FOR DUMMIES®

A Wiley Brand

Semantics For Dummies®, MarkLogic Special Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN: 978-1-119-11220-4 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

Some of the people who helped bring this book to market include the following:

Project Editor: Carrie A. Johnson
Acquisitions Editor: Steve Hayes
Editorial Manager: Rev Mengle
Business Development Representative: Karen Hatten
Special Help: Stephen Buxton, Sara Mazer, Adam Fowler, Rania George, Matt Allen

Table of Contents

Introduction

Semantics is the discipline of deriving meaning from a collection of words or symbols. In computing, it also has something to do with finding meaning in data and refers to a powerful and flexible way of modeling data so that users can have more context to their data than ever before.

About This Book

Semantics For Dummies, MarkLogic Special Edition, explains how databases that incorporate semantic technology can solve problems that traditional databases aren't equipped to solve. Semantics is a way to model linked data (specifically Resource Description Framework — RDF) and forms a graph that can be queried with SPARQL (pronounced *sparkle*). This model is more flexible than the traditional relational data model and very powerful for seeing relationships in the data and discovering new things in the data.

Because this is a *For Dummies* book, you can be sure that it's easy to read and has touches of humor.

Foolish Assumptions

In preparing this book, I've assumed a few things about you:

- You're responsible for increasingly large quantities of data that are starting to arrive in several incompatible forms.
- You may have some familiarity with relational databases, but it seems that situations are arising that aren't a good fit for relational technology.
- You've heard a bit about NoSQL and linked data and are simply curious about what semantics is.

- You want to know how semantics is actually being implemented to create killer apps in the real world.

Icons Used in This Book

You find several icons in the margins of this book. Here's what they mean.

A Tip is a suggestion or a recommendation. It usually points out a quick and easy way to get things done or provides a handy piece of extra information.

Anything that has a Remember icon is something that you want to keep in mind.

A warning alerts you to conditions that require extra care and thinking. For example, you don't want to omit critical steps in evaluating your needs and planning your implementation.

When you see the Technical Stuff icon, it means that the material here is of a technical nature that you may find interesting, but it can be skipped without missing any of the essential concepts being described.

Beyond the Book

You can find additional information beyond what I cover in this book about semantics by visiting the following websites which provide some resources and next steps with MarkLogic Semantics in particular:

- For more information on MarkLogic semantics, visit `www.marklogic.com/what-is-marklogic/features/semantics`.
- To take a free OnDemand course with MarkLogic semantics, visit `mlu.marklogic.com/ondemand`.
- To register for a free instructor-led course on MarkLogic semantics, visit `mlu.marklogic.com/registration`.

Chapter 1

Semantics 101

In This Chapter

- Defining semantics
- Understanding why you need semantics
- Looking at how semantics is gaining momentum

This chapter takes you through the basics of what semantics is and the unique capabilities that it provides, which are particularly valuable in a world that's inundated with data, much of it in unstructured or loosely structured form. You discover the key reasons why semantics exists in the first place, and why semantics is coming to the fore now in today's complex, data-driven world.

Understanding What Semantics Is

As an academic discipline, *semantics* is the study of meaning. It has traditionally been applied to the meaning of words, phrases, and symbols. What a word, phrase, or symbol represents, or stands for, is the domain of semantics. When speaking of computer data, semantics has a more specialized meaning. It deals with entities, both physical and conceptual, and with the relationships between those entities.

Semantics helps make sense of all the information available in today's world by providing a universal framework to describe and link data. It adds contextual meaning around the data so it can be better understood, searched, and shared, enabling

both people and computers to see and discover relationships in the data.

The universal framework for semantics is called the Resource Description Framework (RDF), a standard for modeling data that uses three simple components: a subject, predicate, and object. For this reason, people often just call RDF data *triples,* which are expressed in Figure 1-1.

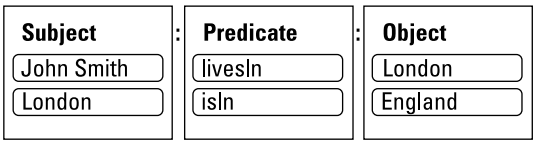

Figure 1-1: Semantic data is expressed as triples.

When triples are linked — the object of one triple being the subject of another triple — they form a graph-like representation of data with nodes and edges that are without hierarchy and are machine readable — easy to share, easy to combine. They're easy to share and combine because each triple is atomic, which means that it can't be broken down any further without destroying the meaning of the fact it represents.

Semantics also has a standard query language, SPARQL (pronounced "sparkle"). SPARQL is a little bit like SQL, but specifically designed for semantic data.

SPARQL is a powerful query language and can be used to search a graph in many different ways. You can even use SPARQL to query for properties that you don't even know exist — a unique capability that distinguishes SPARQL. For example, if you don't know anything about Pluto, you can ask the database to just tell you a bunch of facts about it.

Another thing that SPARQL can do is help infer new facts about the world. You can get an inkling of the power of semantics when you make a simple query, such as "Find people who live in (a place that's in) England." You can combine two facts, such as "John Smith livesIn London" and "London isIn England." Then an inference can be made to create a new fact: "John Smith livesIn England." See Figure 1-2.

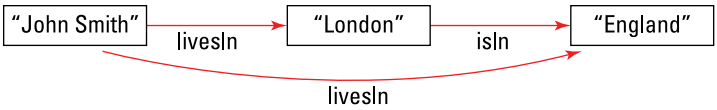

Figure 1-2: Two triples and a third inferred triple.

Without semantics, this example would be easy for a human to infer yet very difficult for a computer to infer. Unlike a relational database, a semantic database, called a *Triple Store*, doesn't require all relationships to be stored explicitly as separate entries. Facts about general knowledge, such as "London isIn England" are often freely available on the Web in the form of triples. Because semantics uses a standard format, organizations can use those freely available facts about the world to enhance their own proprietary data. They can even make inferences by using this combination of data to substantially increase their organizational knowledge.

One example of an open-linked data repository of general knowledge is DBpedia, which organizes the massive amount of information contained in the Wikipedia knowledge base into triples that can be queried by SPARQL. Applying a SPARQL query to DBpedia that essentially asks "Where was Einstein born?" will return the information that "Einstein was born in Germany." For more information on SPARQL, see Chapter 3.

Understanding Why Semantics Exists

Originally, semantics was envisioned as an evolution of the web. It provides a standard format for linking data that isn't too different from the once revolutionary idea of using HTML and HTTP standards to share and link documents on the web. Semantics goes further by providing a universal framework (RDF) to describe and link data, not just documents.

Today, semantics is still often discussed in the context of the "semantic web," but it's more appropriately viewed as a new way of modeling data. At its core, semantics exists because of its amazing ability to add contextual meaning around data so it can be better understood, searched, and shared, enabling

both people and computers to see and discover relationships in the data. This section discusses the key conceptual reasons why semantics was created in the first place and why semantics is being used today by leading organizations around the world.

Storing facts and relationships

To understand the semantic approach to storing facts and relationships, you must first look at how data is currently stored. The traditional approach to storing data in a relational database requires designing the perfect schema that will store the key information in rows and columns.

Consider the example of storing online profiles for a social media site. You'd likely want to create a table with column names for name, age, organization, organization type, role, education, certifications, and so on. For the purposes of optimization, you'd want to normalize the data over a few different tables, creating IDs that serve as pointers between related columns. Pretty soon, you end up with a bunch of different tables set up in your database, and you can start loading data. However, later on, you realize that certain queries run really slow. Some of your queries require a lot of joins and a high volume of data, so you have to de-normalize your data, duplicate it, or build another data model in a data warehouse. Also, because of your data model, it's tough to see how one person is related to another person, and so-on. You also find that you can't easily see how people might be connected through shared commonalities, such as where they went to school or what they're interested in. You also realize that you unintentionally limited how many schools people could list for their education.

Now, contrast that with a semantics approach for modeling parts of the data as semantic relationships. With semantics, it would be very easy to show how one person is related to another. You just state that "John is a friend of Sarah's." When you end up with millions of such relationships, it's not a problem — the database can still quickly retrieve someone's friend list and also show you that person's friends of friends. You could also store other facts such as "John plays soccer" or "Sarah is part of the big data Meetup group" and see who

else is in those groups really easily. And, if a new fact comes up, such as another school that needs to be listed, that's an easy fact to add. Even if that piece of data is from another dataset, as long as it's stored as triples, you can combine that into your own dataset and search it — there are no limitations.

Just about anything that can be modeled as a relationship can be modeled using semantics. It's what semantics is designed for. In fact, large social networks such as Facebook and LinkedIn have used graph databases to model relationships between friends and colleagues (the differences between graph databases and triple stores are covered in Chapter 4). And, as of a few years ago, even Google started enhancing search results by recognizing semantic markup in web pages (specifically, a simplified version of RDF called RDFa).

Providing context for your data

Another problem of modeling data without semantics is the fact that there's no context for understanding data. Traditionally, computers have had a very difficult time understanding context and meaning. Unlike computers, your brain associates different thoughts in order to come up with the "full picture" of something, whether it's an abstract concept or a concrete inference. Applications have a lot more difficulty connecting the dots, particularly with unstructured information. Semantics addresses this challenge, using a standard for modeling data that essentially makes applications smarter.

As an example of the traditional computer's problem with figuring out what users really want, consider the word *cook* — the computer doesn't know whether you mean a chef, the act of cooking, the Cook Islands, Tim Cook, or an imperfectly composed chess puzzle. Even if the computer did know that you meant a chef, it wouldn't know that you would also be interested in the restaurants that the chef works at in a particular city. Semantics addresses this problem of intelligibility by using controlled vocabularies, taxonomies, and ontologies — each of which are designed to describe data so that it can be understood in context. Ontologies are discussed more in depth in Chapter 3.

RDF can provide context at several different levels:

- At the documents and data level, both structured and unstructured data of very different types can be modeled as RDF triples:
  - Relational, or tabular data (a primary key in a relational table can easily be mapped to a URI in a triple store)
  - Entities in free text, such as Albert Einstein is a person, Coca-Cola is a product, Google is a company
  - Events with free text, such as Einstein won the Nobel Prize, Facebook acquired Oculus
  - Document metadata (categories, author, publication date, source)
  - The provenance of a data item (Where did it come from? How reliable is it?)
  - Modeling reference data such as the name of the company that has the stock ticker AAPL. Who is that company's CEO?
- At the domain level, taxonomies and ontologies for specific domains can provide context:
  - A pharmaceutical company's drug ontology
  - SNOMED CT clinical healthcare terminology
  - Dublin Core Metadata Initiative for media and publishing resources
  - FIBO (Financial Industry Business Ontology)
- At the level of the world at large, billions of facts are shared freely:
  - DBpedia translates the information in Wikipedia into triples such as "Einstein was born in Germany" and "Ireland's currency is the Euro." DBpedia contains billions of pieces of information as RDF triples.
  - GeoNames contains geographical data such as "Doha is the capital of Qatar" and "Doha is located at 25.2887° N, 51.5333° E."
  - Linked Open Data is a collection of thousands of freely available data sets that are all interconnected.

Avoiding the creation of "walled gardens"

By their very nature, applications written to process data stored in traditional relational databases focus on the immediate task at hand. If you later come up with a different task, you may have to create a new database structure, even if the data you're dealing with is essentially the same as that dealt with by the first application. Or, what happens when your company merges with another and you have two major sets of customer records? How would you create a unified view?

Walled gardens of data are common, and they prevent the data from being used for anything beyond each application's original design. Traditional database systems aren't equipped to deal with multiple different kinds of data, which is why billions of dollars are spent every year on data integration projects, and yet often just result in more data silos.

The solution to this problem is to link data together so it can be searched holistically. Semantics deals with the relationships between data, making it an ideal tool to link and search across structured and unstructured data. This is particularly useful in creating sophisticated queries that span multiple data sets. Triples can be used to map different field names from one dataset to another. So, a single query can bring in all the information about a particular customer, for example, because "cust123" is the same as "cust_id_456".

Speeding up application development

The way to speed up any task is to do more of it automatically and less of it manually. The examples in the preceding section show how data integration projects can go much faster. Another way that semantics speeds up application development is by making complex queries much easier. For example, consider the plight of the application developer who must write an application that gives users answers to such requests as, "Show me all the researchers who currently live in the United States who cite another researcher who cites John Smith, and return their names and all their publications."

In the absence of semantics, each of the data sources involved must be treated a different way and the various results must somehow be integrated in a way that produces the desired answer. This is a job for a highly skilled programmer working for many hours while consuming many pizzas and downing numerous 2-liter bottles of soda. Or, a semantic system could do it in a matter of seconds, without either the pizza or the soda. Triples of the form "X sameAs Y" enable queries and inferences to be made across boundaries created by data of different types from different sources.

Semantics may seem hard to learn, but it's actually pretty straightforward when you get the hang of it and can make your life easier.

Visualizing dense data

If you have triples, you can visualize them in a user interface to give users the ability to see the relationships. Visualization often leads to users seeing new patterns they didn't see before, and it provides new ways of analyzing data. Some examples of tools and libraries commonly used for semantic data include

- Open source solutions such as "vis.js" (`http://visjs.org`) or d3 (`http://d3js.org`)
- Commercial options such as KeyLines (`http://keylines.com`)

Production applications are using visualization tools to analyze big data for the purposes of uncovering anomalies in cybersecurity, improving military intelligence to catch bad guys, and many other industries and use cases.

The visualization in Figure 1-3 was built using a MarkLogic REST service and the `visjs.org` library for some dense health data.

Vis.js handles a lot of nodes and dynamically finds the best arrangement and has an easy-to-use API. You can assign single-click, double-click, and other events easily to nodes, edges, and whitespace. Users can navigate the graph by double-clicking nodes, querying for everything connected to

that node, or clearing the stage and re-centering the results. Since the data in this graph is really dense, the graph is expanded from just nodes and edges to nodes and "virtual relationship nodes" and edges.

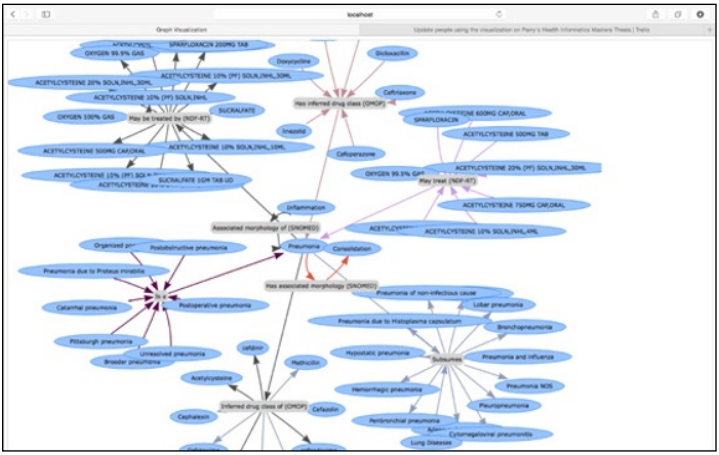

Figure 1-3: An example of a graph visualization of dense healthcare data.

Seeing Why Semantics Is Gaining Traction Now

Organizations are finding that the problems with big data — its volume, velocity, and veracity — are only going to get worse in the coming years. In 2013, there were about 4.4 zettabytes of data in the world. By 2020, it's estimated that there will be over 44 zettabytes of data. That's about 6,000 Gigabytes for every person on Earth.

While the increasing volume of data is impressive, the bigger challenge is the variety of data. Most organizations have a wide variety of tools to manage their structured data, but even that task has become cumbersome with the constant ETL processes and complex data modeling. And, the task of storing and managing the other 80 percent of the data, which is unstructured, is even more challenging because traditional databases and tools are designed for data that fits a pre-determined schema, not unstructured documents and graph data. The unfortunate

result is that IT departments are forced to spend the majority of their time managing the critical flows of data with a polyglot of various tools. And application developers are stuck trying to shoehorn heterogeneous data into legacy data models rather than having the freedom to choose the model that works best for their current data and applications.

Semantic technology has evolved to address today's challenges by providing the ability to more easily integrate heterogeneous data. And, from the outset of new projects, it gives application developers a powerful option they can leverage when they want to model relationships and bring context into their applications. With semantics, the variety of big data becomes an opportunity rather than a problem because data can be easily mapped and modeled. Data is given context within the domain that it lives in and makes it easy to model the inherent relationships between entities. In this way, applications are made smarter and users can simply find and explore what they're looking for faster and easier.

With the ability to store documents, data, and triples all in the same database, MarkLogic is uniquely designed to more easily integrate data from a variety of sources in its myriad of forms. And, it makes it possible for developers to create smarter applications that generate better answers to harder questions.

Tracing the history of semantics

The idea of modeling knowledge as a semantic network was developed in the early 1960s by cognitive scientist Allan M. Collins, linguist M. Ross Quillian, and psychologist Elizabeth F. Loftus. When the concepts described in the publications of these researchers were applied to the Internet, they took the form of hyperlinked human-readable Web pages. Machine-readable metadata about pages and how they related to each other formed the links. This structure enabled automated agents to traverse the web and perform tasks for users. Tim Berners-Lee became a champion of this new model and, in an article he co-authored with Jim Hendler, named it "The Semantic Web." Previously he had invented the World Wide Web and become the Director of the World Wide Web Consortium (W3C), which, among other things, oversees the development of proposed Semantic Web standards.

Tim Berners-Lee's vision was to use linked open data to make connections between data items that weren't previously known to be related. He envisioned this linked data as a graph, as shown in Figure 1-4.

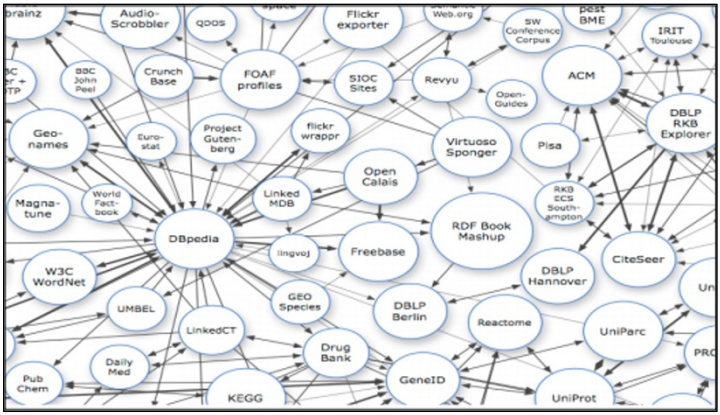

Figure 1-4: A graph of linked open data.

RDF soon became the commonly accepted data format for semantic data. Early pioneers of semantics found that using RDF to model linked data helped liberate data from the containers it came in, making it available for a variety of automated processes. The W3C publishes the standards that define RDF. RDF is based on using URIs to look up and describe resources.

URLs, URNs, URIs, and IRIs

When reading about semantics, it won't be long before you run across the acronym URI. What the heck is that? It looks like it might be similar to URL. I know what a URL is: it is the address of a Web page. When I feed my Web browser a URL it takes me to the corresponding Web page. What does a URI do?

The letters URL stand for *Uniform Resource Locator*. The owner of a domain on the Web controls the directory structure as well as the file names used to store resources in that domain. The URL locates those resources that happen to be Web pages. A domain may also have resources that are not Web pages.

(continued)

(continued)

For example, the resource might be an item identified by name. The name would be a *Universal Resource Name* (URN), and have the form of the following example:

urn:isbn:978-1-118-90574-6

This is the identifier for the book *NoSQL For Dummies* by Adam Fowler.

After URNs were defined, the more general term *Universal Resource Identifier* (URI) was invented to refer to both URLs and URNs. If you have a URL, there's an expectation that if you follow that URL (using a browser), you get to a page. With a URI, which is an identifier, not a locator, there's no such expectation. The URI may also be a URL — there may be a page at the end of it — but it may well just be an identifier, which is a way to distinguish between this thing and any other. Certainly the "syntax police" are fine with calling a URI a URL. But when I talk about triples, the subject and object must be an IRI.

Oh, there's one more thing. The ASCII character set used to specify a URI contains only 256 characters. This isn't enough to hold all the characters needed for Cyrillic alphabets or Chinese characters, so the Universal Resource Identifier has been expanded to accommodate larger character sets, and is now designated the *Internationalized Resource Identifier* (IRI). A URL is also a URI, which is also an IRI. URNs are used only rarely, so most times URL, URI, and IRI mean the same thing. Those most concerned with being politically correct use IRI.

Handling big (complex) data

Business systems, scientific instruments, and now the myriad of sensors that are an integral part of the Internet of Things (IoT) are creating a mind-blowing amount of data every second in a variety of forms. Adding to the problem is the fact that the hardware processing the data is also evolving at an exponential rate. Trying to handle this torrent of data along with on-the-fly hardware upgrades is like trying to drink from a fire hose. The tools of the past are ill-equipped to deal with the challenge. Not only is there more data than ever before, and it is arriving faster than ever before, but it comes in a wider variety of forms than ever before. Semantic technologies offers the best hope of managing this exponentially growing resource by providing a consistent and flexible way of managing all of the data — and all of the inherent connections within the data.

Implementations of semantic systems have a range of capabilities, but some are able to handle truly huge amounts of data. Using a combination of indexes on a triple store and a triple cache, some products are able to deliver truly speedy query responses — across hundreds of billions or even trillions of triples.

Linking documents, data, and triples

It's easy to fall into the trap of thinking big data just means storing more of the same thing, but that isn't the case. Big data means large volumes of many different, changing things. You may have some tabular data coming from a relational database or old mainframe, some JSON or XML documents, and maybe even a few billion triples. Handling all of this data quickly, and avoiding the resulting mess of various tools and transformation, is easier said than done.

Or, you may be looking at it from the other way around: You're trying to build a big data application, and you have some data you want to store in a highly structured tabular format and other data that's best modeled as semantic relationships. But, the problem is that your infrastructure dictates that you use a standard relational database. So, you then make the case for purchasing a graph database or triple store. And then a search engine. Oh, and maybe an ETL tool to bring some old reference data into your application. There is a better way!

The truth is, modern enterprises need flexibility to handle all their data. They need to have a way to store and manage documents, data, and triples in one place, where they can be easily and quickly managed, searched, and combined.

MarkLogic is an Enterprise NoSQL platform that *can* store and query a combination of documents, data, and triples. With a single platform, users have flexibility in choosing the data model or mix of models that works best to store their data and provides the ability to query across everything holistically. With MarkLogic, triples can be embedded in documents, triples can refer to documents, or triples can connect

documents. And, in MarkLogic, users can choose how to query the data, using either JavaScript, XQuery, or SPARQL — or even a combination of languages. In the real world, having this flexibility is critical.

Choose a vendor, such as MarkLogic, that can store and query documents, data, and triples in the same database.

Chapter 2

Looking at Semantics in Action

In This Chapter

- Modeling data using semantics for more flexibility and context
- Solving data challenges using semantics at leading organizations

Semantics is being used right now in the real world in a wide variety of applications. In this chapter, I discuss how semantics is being used at a practical level by a few very different organizations, showing how semantics is a powerful and versatile technology used to achieve things that were otherwise impossible.

Discovering What You Can Do With Semantics

Simply put, semantics makes applications smarter. This is because in any collection of data items, in addition to the facts that are explicitly stated, there is considerable additional information that is present, but unrecognized. Linking the data items to one another unlocks that hidden information and makes it available to queries made into the data store.

Building and traversing graphs

You probably have a pretty solid idea in your mind as to what a graph is. You have seen plenty of graphs all your life. One example would be a price graph of your favorite stock.

Another would be a graph that you add a dot to every morning after weighing yourself. Those are graphs all right, but they aren't the kind of graphs that I'm talking about when dealing with databases, or with pure mathematics for that matter.

In mathematics, the word *graph* refers to a very well-defined structure. According to the strict mathematical definition, a graph is a collection of objects, called nodes or vertices, which are connected to each other by lines called edges or arcs. An arc represents a relationship between two nodes.

A collection of triples that deal with items that are related to each other is also called a graph, so you build a graph by creating a collection of linked triples. An entity (or item) that's the object of one triple may be the subject of another. The predicates that connect subjects with objects create a web of interrelationships.

Figure 2-1 shows the simplest possible graph, representing a single triple. David Bowie and London are nodes and birthPlace is an arc.

Figure 2-1: A simple single triple graph.

In this graph of a triple, *David Bowie* is the subject, *London* is the object, and *birthPlace* is the predicate that connects the subject node to the object node. It is a flat graph. There is no hierarchy, with no single node of the graph being any more special than any other. The subject and object of the triple comprise the nodes of the graph, and the predicate of the triple is the arc or edge of the graph.

You can expand the graph by adding more triples, as shown in Figure 2-2.

You now have two facts, and they both relate to David Bowie. With semantics, you can use the same IRI to represent the same David Bowie, so the graph might look like Figure 2-3.

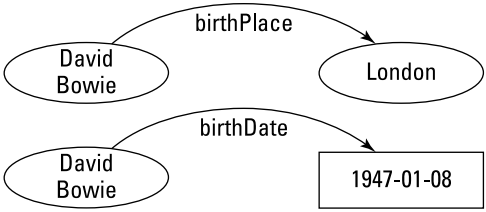

Figure 2-2: A more complex graph that uses multiple triples.

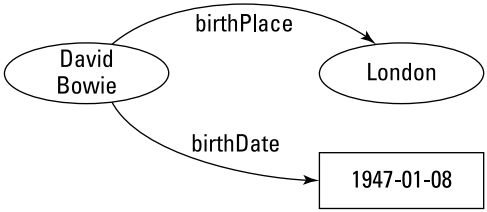

Figure 2-3: The combined data in a graph.

By adding more triples, you get a more complex graph. The more complex the graph, the more likely that it contains knowledge that goes beyond what's explicitly stated in the triples that make up the graph. Figure 2-4 shows one possible example of this idea.

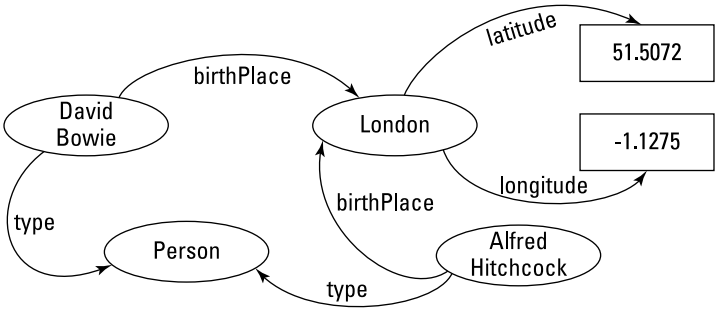

Figure 2-4: As more triples are added, additional information can be inferred.

Although it isn't explicitly stated with a triple, Figure 2-4 shows that semantics enables a David Bowie fan who is also a geography geek to infer that since David Bowie was born

in London and London is at latitude 51.5072° N, then David Bowie was born at latitude 51.5072° N, a new fact. We can also infer that David Bowie was born in the same place as Alfred Hitchcock.

Sharing and searching facts

A lot of information, such as the latitude and longitude of London, is freely available in open source data stores. Some of these data stores contain hundreds of billions of triples. This can be a tremendous benefit when such data is added to the proprietary data that an organization may possess. The chance for inferring new, potentially useful facts expands exponentially.

The ability to store hundreds of billions of facts is only valuable if you can quickly retrieve those facts later on. Retrieval entails search. It must be possible to search through the data in a data store and return a result in a reasonable time. Triple store databases deliver good response times by having indexes of their content.

Discovering hidden nuggets

Making connections between huge numbers of facts enables you to discover things that you may have ordinarily missed. You can traverse and analyze a graph to see how one thing is connected to another thing, even if it is separated by many degrees. An example of how this looks in practice can be seen in Figure 2-5, an application built using MarkLogic that shows how people and places can be connected. Applications like this can be really helpful in the world of intelligence to find out how one bad guy may be connected to other bad guys, and how they are connected through payments, where and when they lived in a certain location, and anything else they may share in common.

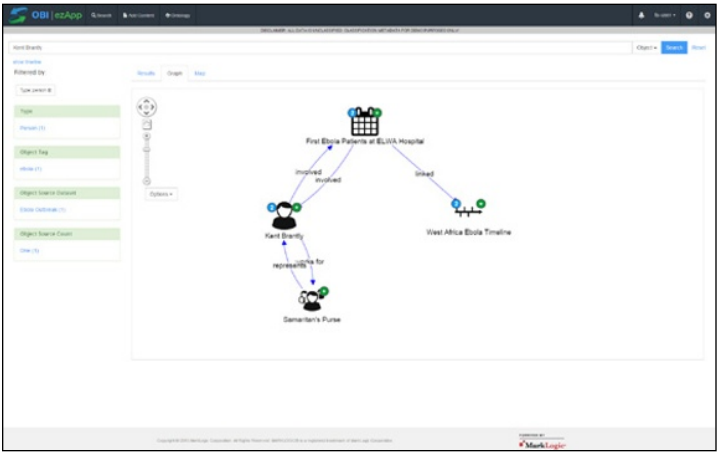

Figure 2-5: One view in a semantics application that shows how people, places, and things can be connected.

Highlighting Successful Examples in the Real World

If semantics is a new concept to you, you may be surprised to discover that it is already in use in a variety of contexts by very different organizations to perform important functions. In the sections below, I discuss some of those functions and the organizations that are currently making use of semantics to perform them.

Making search intelligent

One of the most common uses of semantic technology is intelligent search. *Intelligent search* is an enhancement to the familiar search capability you would find with enterprise search technology. In addition to indexing your data to make it easy and fast to retrieve, intelligent search adds a semantics component that helps deliver better answers to harder questions. The examples provided here have to do with searches across documents, data, and triples to get insights from all your data.

Television broadcasting company

A large television broadcasting company launched an intelligent search application as part of an anniversary celebration for one of its popular television shows. A three-hour special was televised featuring many present and past cast members of the show, and it brought in over 20 million viewers. However, the most enduring part of the celebration might have just been the mobile app that was released alongside the show to deliver on-demand content to fans.

The mobile app was the first enterprise-class mobile app to use semantics. It uses a semantic data model to give fans of the show the ability to quickly and easily find favorite videos from over thousands of videos that were aired over the course of a 40-year time period. The app also includes a recommendation engine, driven by semantics, that adapts to a fan's preferences over time.

When fans first download the app and log on, they're asked to authenticate using Facebook and then they're asked which era of the show they love most. Because the app uses semantics to tie each video to an era, it can easily deliver a package of videos that fans can watch or swipe through. The app logs each one of those interactions and then changes future recommendations based on each fans interactions. Semantics is also used to quickly filter out videos that users have already viewed from its recommendations.

Semantics is what gives fans the ability to truly explore content rather than just watch content. Traditionally, an app may provide a listing of content that you can scroll through and pick what you want to watch. Sophisticated apps, such as Netflix, have improved on this model by providing a listing along with a model for predicting what content you may like and listing that as well. But, even with Netflix, you still won't be able to navigate across the relationships in the data. For example, you would still need to visit another source in order to find out what other movies an actor was in.

With the app developed by this large broadcasting company, fans can pull up a video, see and click on the actors and characters and related tags on that video, and even see photos of what the actor looked like in that era. See Figure 2-6.

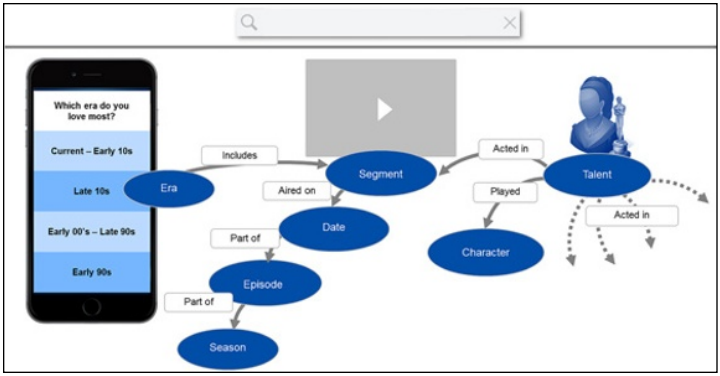

Figure 2-6: A semantics-driven search.

It's semantics that allows fans to explore content in new ways that weren't possible before. Instead of dictating a linear user experience in which fans generally know what they're looking for, semantics opens up to discovery, to allow fans to more easily find content they didn't even know existed.

All the critical data in the mobile app is stored natively as RDF triples. Powered by MarkLogic, the app is launched from the cloud and leverages the elasticity of MarkLogic to scale up to meet a spike in demand and to quickly scale back down when the demand dissipates.

The American Psychological Association (APA)

With nearly 130,000 members, The American Psychological Association (APA) is the world's largest association of psychologists. APA manages and indexes both structured and unstructured data for psychology. This includes articles, books, journals, dissertations, videos, tests, measures, and more. One of its key products is a subscription database called PsychInfo, which includes over 73 million cited references.

APA's main function is to deliver information and knowledge to a mostly academic audience. It needed a technology to deliver that information easily and quickly. APA had been using MarkLogic for a number of years, but the introduction of semantics enabled the association to extend its document data model with the addition of RDF.

Semantics helped the APA create an advanced semantics search application analytics application for its subscribers that allows them to visualize complex relationships among authors, affiliations, sponsors, subject areas, and more. APA manages all the relationships using a customized ontology they built that defines all the relationships in the data. Users can then do searches across the data. For example, they can search for an author who had a citation listed, and then also find who cited that author, who cited the author who cited that author, and so-on. Figure 2-7 shows another example in which an analyst is looking at the frequency of a certain term, in this case "divorce," is used in various journals over a 30-year period. By analyzing the relationships, this analysis is easily done quickly at scale.

Figure 2-7: The frequency of a term.

APA is also leveraging semantics to completely change the way users search for content and for the authors who write that content. For example, when a search brings up a page about a particular author, it displays a picture of the author, a list of all the author's publications, topics the author studies, the most common co-authors, sponsors, and the frequency of publication over time. This could all be tied very easily to a user's search for either a topic the author wrote about or the author's name. Even more sophisticated searches are possible, such as "Show me more articles like this one." Other examples are "Give me a list of journals that discuss this

particular topic the most over time," or "Who are the most common sponsors and co-authors for this topic."

Mitchell 1

Mitchell 1 has been providing detailed information about both foreign and domestic cars for over 90 years. As cars have become more complex, the job of the neighborhood auto repair shop has become increasingly difficult. Repair manuals have increased in thickness, and additional information has been made available on CDs and DVDs. It's often difficult for a repair technician to know where to turn for needed information. Mitchell 1 has addressed this problem by putting all the information about all the cars in one place with MarkLogic doing the heavy lifting under the covers. The system, called ProDemand, has over 100 million documents in it, with data streaming in from 28 different manufacturers. Whatever problem a technician is facing, someone else has faced it before, and the solution is now a few keystrokes away with Mitchell 1's search tool.

Another problem that Mitchell 1 faced was having so many different parts that were each called by the same name. For example, technicians may call a hundred different parts a "gasket," so when it comes to searching for a "gasket," it's not clear at all what part is being referred to. Another problem is that it isn't usually clear how one part may be related to other parts. A technician would obviously want to know how one part of a car is related to another part, whether it's seeing how a "conditioner compressor" is related to the engine cooling system, or if he wants to know what part of the engine cooling system is composed of if starting from a higher level in the hierarchy.

With automotive data, however, there's no agreement about what to name things. Given that there are 196,000 unique vehicles to keep track of, there's plenty to disagree on. Mitchell 1 had the task of either managing four completely different parts vocabularies that each had a different approach to naming things, or to try and create a more unified core parts vocabulary. By using semantics, the company was able to keep each vocabulary separate but allow users to see how each vocabulary relates to the other. See Figure 2-8. In this way, a user can easily search for "gasket" and see what the core term is and how that term, or part, relates to the rest of the engine as a whole.

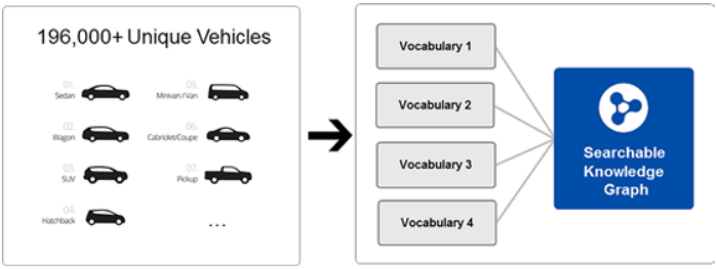

Figure 2-8: Mitchell 1 uses semantics to create a knowledge graph for car parts.

Mitchell 1 expects to use the knowledge graph to present its customers with a fully integrated view into everything the company knows about a specific vehicle or element of that vehicle.

Simpler data integration

Epinomy, shown in Figure 2-9, is an application built on top of MarkLogic that makes it possible to load multiple data feeds, extract and tag meaningful information from the data, and search across it. Epinomy can handle time series data, tabular data, multimedia, office documents, web pages, and other kinds of feeds. It can ingest them all and use semantics to pull more meaning from them. The ability to handle data from disparate sources is especially valuable in organizations facing interoperability challenges due to mergers and acquisitions.

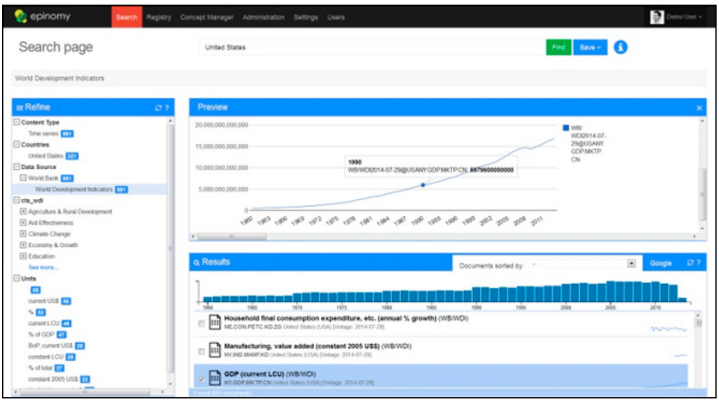

Figure 2-9: Epinomy uses semantics to enable users to search across disparate sets of world economic data.

One of the big problems that Epinomy addresses is that of integrating data from different sources in which there are different names for the same thing and the same name for different things. Even more challenging is the naming of things that are similar but not identical to each other. For example, Epinomy is used to manage data for multiple economic indicators. It's difficult to combine various sources for economic data with related terms such as "Euro zone," "European Union," "Europe OECD," and "Europe."

Epinomy makes it easy to ingest and transform various data types, as shown in the user interface in Figure 2-10, and have that data stored as XML, with triples being used to supplement the raw data. For example, there may be a time range in which a particular fact is true, such as when a country is a member of the Organization for Economic Co-operation and Development (OECD) or the geographical boundaries of a country changed at some point in time. Triples provide a perfect data model to manage those changes. In Epinomy, triples are actually embedded within XML documents, an approach known as working with "embedded triples." Doing so reduces the computational load when analyzing the data. This approach was possible because the development team chose MarkLogic, which can store and index both documents and triples in an integrated fashion.

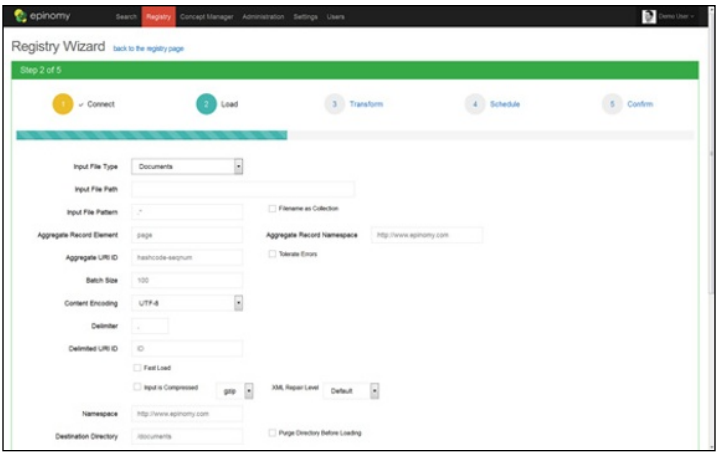

Figure 2-10: Epinomy's intuitive user interface.

Dynamic semantic publishing (DSP)

Hosting the 2012 Summer Olympics was a very big deal for the United Kingdom, which wanted to show the world that the UK knows how to put on a world-class sporting extravaganza. Equally on the spot to show that its technology was on the leading edge of what's possible, the British Broadcasting Company (BBC) wanted to deliver the Olympics to a worldwide audience in real time, along with appropriate commentary, statistics, and relevant facts about the events and participants.

BBC brass had earlier come to the conclusion that the static publishing solution that it had been using since the 1990s, based on relational database technology, wouldn't be able to carry the load it was expecting, nor would it be able to provide the comprehensive and highly responsive coverage that would be required.

The BBC decided to move from a relational model and static content publishing framework to a fully dynamic semantic publishing (DSP) architecture. DSP uses linked data technology to automate the aggregation, publishing, and repurposing of interrelated items.

DSP was necessary, because for the 2012 Olympics, the BBC needed to publish content across 2,000 micro-sites, including pages for over 10,000 athletes, 200 teams, 400 disciplines, dozens of venues. And, it had to do this without adding any more content editors. By using DSP, the BBC could make an update to one piece of content, or fact, and have that change propagate across any page that referenced it.

The DSP infrastructure was built using MarkLogic's Enterprise NoSQL database platform and also an Ontotext triple store. Throughout the competitions, heterogeneous data (for example, IOC data, tweets, images, and video) continuously and dynamically streamed into the Olympic website from a wide range of organizations, channels, and journalists. This unified, interactive repository was able to handle the crush of over 25,000 transactions per second, creating custom experiences for users who made 106 million requests for live and on-demand video. Check out Figure 2-11.

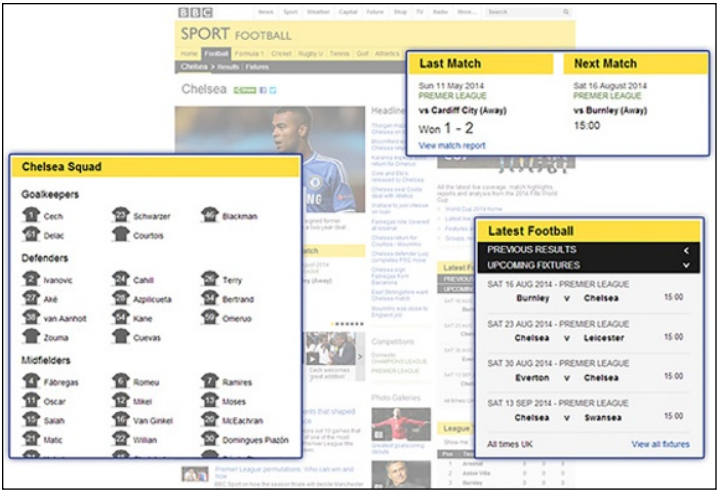

Figure 2-11: Data is updated in one place, and RDF is used to link the data together so it can be easily found and re-purposed.

By using MarkLogic, the BBC was able to accomplish the following:

- 106 million requests for BBC Olympic video content
- 55 million global browsers across the games
- 2.8 Petabytes of data on a single day
- A daily record of 7.1 million UK browsers

The combination of the NoSQL content store and the triple store provided an unparalleled level of automation and dynamic content delivery. The BBC has remained committed to using semantics and is also implementing semantics in other applications as well.

Semantic metadata hub

You've heard the expression, "Content is king." This maxim is maximally true for a media company whose stock in trade is motion pictures, video games, comic books, or TV shows. Product titles, characters, distribution rights, and technical information are all vitally important but are likely to be maintained in information silos that are separate from each other because they were created or acquired at different times,

under different circumstances. Such disconnected systems lack common standards and thus are difficult to integrate into any kind of an overall strategy.

The solution to this problem is to establish a semantic metadata hub. Ideally, it would consist of a flexible, document-oriented, schema-agnostic database that has semantics integrated in. Having everything in one place and accessible by one mechanism enables rapid response to emerging business challenges or changes in the competitive landscape. One major entertainment company addressed this challenge by consolidating their diverse and separate assets into a semantic metadata hub. They built the metadata hub using MarkLogic for storing and searching the data, and also Smartlogic for classification, publishing, ontology management, and semantic enrichment functions.

The metadata hub works by ingesting data from multiple diverse sources and then providing a natural language search interface for users to query any of the assets or metadata in the system, without losing the ability to see where the data came from. Because the data model is driven by semantics, users can see the relationships in the data, easily visualizing how a movie had a certain star playing in it, and that she played a certain character in that movie, and the character was from a certain place, and that the movie was animated. Figure 2-12 gives you an illustration of this concept.

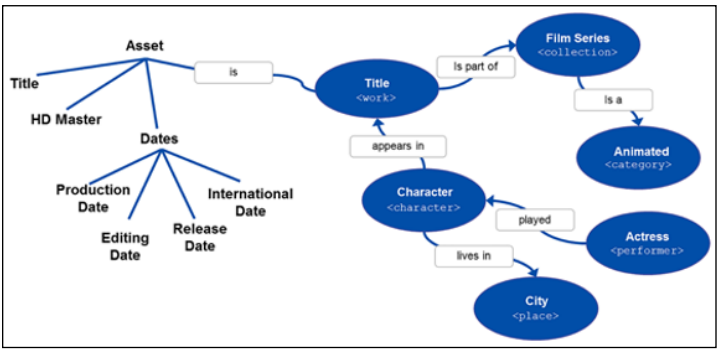

Figure 2-12: The data model for the semantic hub uses a combination of XML documents and RDF triples to provide a complete picture of each media asset.

Object Based Intelligence (OBI)

Military and government intelligence agencies are dealing with threat environments that could lead to very serious consequences if all the multiple sources of information that they use can't be organized in a manner that gives them real-time insight into a developing situation. Actionable intelligence delivered in a timely manner, based on multiple, diverse sources, is what is needed. Intelligence agencies are tasked with dealing with Donald Rumsfeld's famous triad of known knowns, known unknowns, and unknown unknowns. Having a good handle on the known knowns helps to provide some insight into the other two, much harder threat categories. Figure 2-13 shows an application built on MarkLogic, called OBI EZApp, that demonstrates how OBI works in practice.

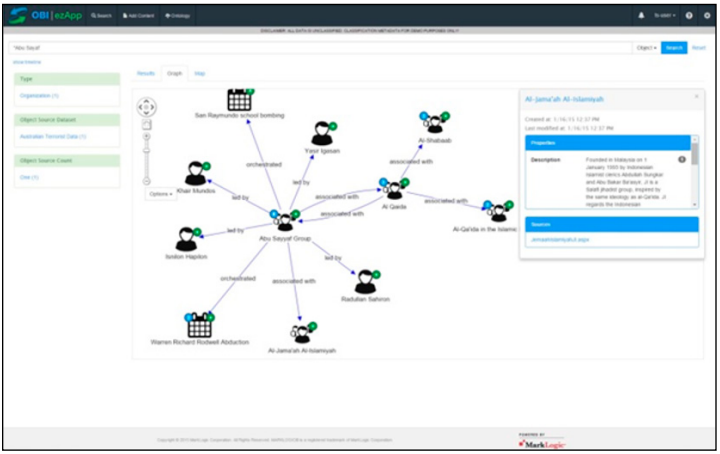

Figure 2-13: OBI EZApp, an application built on MarkLogic, shows how people, places and things can be related by using semantics.

Object Based Intelligence (OBI) is based on the premise that intelligence should be organized around objects of interest, where objects of interest could be people, groups, vehicles, buildings, or other known things.

The object types and attributes are defined by ontologies, and include:

- Data, including multiple attributes and relationships
- Context, including the history, location, and associated community of interest
- Value level metadata, including value-level security, provenance and pedigree (trustworthiness) and time validity or periodicity
- Content and knowledge, including semantic facts, attachments, imagery, text, and links

The OBI framework that overlays the data allows operators to view and manipulate the data and semantic relationships, and maintain dynamic attribute lists so a given piece of data is available to multiple objects. Any piece of data can also be updated or linked graphically, in real-time. And, fine-grained security controls ensure people see data according to their role.

Ultimately, the OBI architecture is about analysts and decision makers, their information needs, and the management of complex knowledge domains. IT answers the challenge of unruly and exploding information volume and varieties by providing tools and a data model capable of creating, storing, enhancing, and disseminating high value analysis products. It places intelligence in context by answering who, when, what, and where.

Compliance

If you want high stakes and adrenaline rush action, the place to go isn't the World Championship of Poker, the stock market, or even the derivatives market. It is the energy trading market. Practically any type of energy commodity for which sufficient supply and demand exist can be traded, such as electricity, CO_2 allowances, gas, coal, and oil — and traders can make a significant amount of money hedging and speculating on the changing flows of energy, hoping that they are on the right side of a volatile market.

Because electric power generation, transmission, and distribution is so critical to society, it's heavily regulated by governments, including energy trading. There has also been

generally increasing regulation following the world financial crisis. The Dodd-Frank Act and Volcker Rule, along with other industry regulation, have been implemented to keep speculative trading in check, and as a result any company involved in trading must be sure they are following all the guidelines to the letter.

One of the larger companies in the energy trading business was struggling with keeping up with all the new regulation using its legacy system. They stored records of transactions, including the size of the deal and the traders involved, onto disk, where Microsoft Access databases and Excel spreadsheets were populated. Printouts of the data were then analyzed by hand. Other forms of data, that were relevant, but not captured, disappeared into the ether. Naturally, there were huge gaps in regulatory compliance, and it was only the fact that the responsible regulatory agency was not adequately staffed to check everybody all the time that kept most of the players out of trouble.

Understanding that what you don't know *can* hurt you, this company took the proactive step of putting not only their transaction tracking data, but also market data, weather data, and trade communications into a MarkLogic data store (this is shown in Figure 2-14).

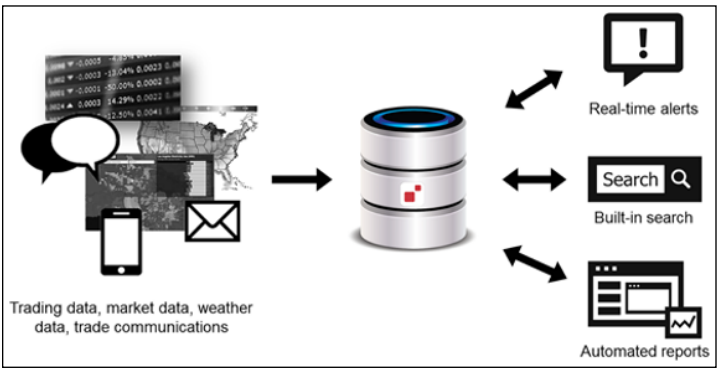

Figure 2-14: An energy trading company can integrate disparate data and use semantics to better comply with industry regulations.

Now, the company is working on using semantics to make tagging and analyzing unstructured data part of the system as well.

Using semantics, as shown in Figure 2-15, the company can tie together a trader's name, role, and company to his or her IM handle. Then, with this basic format, the system can start bringing in IMs, emails, transcripts, and so on, and start connecting that to the more structured data.

Figure 2-15: Semantics can be used for more intelligent regulatory compliance of financial trading.

By using the new semantic data model, an auditor could analyze instant messages between traders to see if they are complying with industry regulations. As they search across IMs, for example, they could mark it up saying, "I haven't seen that word or phrase before, I'm not sure what that means. But, let me note that because it may be important later on."

This process would be creating a new triple on the fly to relate that certain phrase to a particular trader, and maybe to something else, such as a trading strategy being used at the time, or just to a particular trade or trader. The auditor could then create a real-time alert to get informed if that same word or phrase gets used again, helping uncover any fraudulent activity in a smarter, more automated fashion.

Chapter 3

Data Modeling

In This Chapter

- Modeling semantic data
- Deriving more meaning from your data with ontologies
- Querying semantic data
- Finding ready-made sources of triples
- Using tools to create triples and manage ontologies

This chapter discusses how modeling semantic data differs from modeling traditional relational data, the place of ontologies and of the SPARQL query language. Inferencing is covered, as well as combination queries and the many tools that are available to build applications that use semantics to infer additional information that is implicit in the data that is explicitly present in a database.

How to Model Semantic Data

It does no good to possess a large quantity of data, if the particular piece of it that you want at a particular time cannot be easily and quickly retrieved, without also retrieving a lot of data that you don't want at this time. To achieve the goal of easy and quick retrieval of desired information, a number of models have been used over the course of history. For the past several decades the predominant data model has been the relational model. It has worked well, but recently people have discovered situations in which the relational model is not as well-suited, and that other approaches to modeling data have evolved to better handle the structure, relationships, and context of today's data.

The RDF data model

RDF stands for Resource Description Framework, a framework for representing information in the web. It is a standard maintained by the W3C (`www.w3c.org`). There is a data model attached to the framework that describes an abstract syntax that links all RDF-based languages and specifications. This abstract syntax has two key data structures: RDF graphs and RDF datasets. RDF graphs are sets of subject-predicate-object triples. RDF datasets are organized collections of RDF graphs.

An RDF triple is an assertion that says that some relationship, indicated by the predicate, holds between the subject and object. Such an assertion is known as an RDF statement.

A stand-alone RDF triple looks like what you see below, which states the fact that Aaron Rodgers is on the team named the Green Bay Packers:

```
<http://dbpedia.org/resource/Aaron_Rodgers>
<http://dbpedia.org/ontology/team>
<http://dbpedia.org/resource/Green_Bay_Packers> .
```

This RDF example uses a particular kind of RDF serialization called *Turtle,* which stands for Terse RDF Triple Language serialization. Turtle is very simple and uses just a grouping of three IRIs to represent the triple. There are also other serialization formats as well, including JSON and XML. Regardless of the format, the subject, predicate, and object are all clearly represented.

Ontologies

An ontology is a formal definition of the types, properties, and interrelationships of the entities that exist in some domain of discourse. It provides a shared vocabulary that can be used to model a domain. The objective of an ontology is to describe some domain, classifying and categorizing the elements contained within it. If I want to say that "Henley" is a subclass of "Shirt", it's the ontology that gives me my vocabulary for saying that. It gives me the IRI that denotes "subclass" as well as the formal definition of a subclass.

OWL, short for Web Ontology Language (Go figure!?), is a knowledge representation language for authoring ontologies.

I suppose OWL is easier to pronounce than WOL, lends itself better to the construction of cute logos, and as a bonus, implies wisdom. OWL is characterized by formal semantics built on RDF and the RDF Schema (RDFS). The RDFS specification enables you to specify classes and properties, as well as metadata about those classes and properties. From this metadata, you can infer new facts about your data.

The data described by an ontology can be interpreted as a set of entities that serve as the subjects and objects of triples in an RDF graph, and a set of assertions about those individuals that serve as the predicates of that graph. These predicates can represent properties of the entities. An ontology also contains a set of axioms that place constraints on the set of entities, which are called *classes,* and on the types of relationships permitted between those entities, which are the *predicates.* These axioms enable systems to infer new information, based on the data that the graph already contains.

Taxonomies versus ontologies

A *controlled vocabulary* is just a list of terms and definitions that are formally defined. A *taxonomy* is a collection of controlled vocabulary terms that show a bit more about how they're related. They're generally seen as parent-child hierarchies. The easiest example that you're probably already familiar with is from your biology class when you learned about "Kingdom – Phylum – Class – Order – Family – Genus – Species". Another examples is "Earth – North America – United States – California – San Carlos." Taxonomies are limited, though, because they don't say anything about the type of relationship between each thing. For example, a computer wouldn't know that San Carlos is a city in California. And, how would you relate the city of San Carlos to the zip codes it contains or its geographical coordinates? That's where ontologies come in.

Ontologies are a bit more complex than taxonomies. They provide strict semantic models for portions of the world so hierarchies are given meaning. Ontologies are generally expressed as triples and rely on ontology languages such as Resource Description Framework Schema (RDFS) or Web Ontology Language (OWL).

RDF is based on IRIs so that a particular cook (or a word that means a particular "cook" definition) can be represented as an IRI. The last "I" is "identifier," which means it identifies exactly one thing.

(continued)

(continued)

An ontology provides a controlled vocabulary, so it tells us what IRIs to use for particular things (entities) and relationships (predicates). And an IRI defines what those predicates mean. So, for example, there's a formal definition of a subclass — a class A is a subclass of a class B if everything that is in A is also in B. Ontologies generally also include categories and hierarchies of terms.

By using this example, you could say how San Carlos is related to other things, stating that it's a city in California, and it's located at 37.4955° N, 122.2668° W.

In addition to helping build better navigation and search experiences, ontologies are also helpful in publishing more relevant content and making sense of metadata.

How to Query Semantic Data

When data is stored in triples, in what is called a triple store, there must be some way to search and query just the information you want. There are several ways to pull desired data from an RDF graph in a triple store. The most commonly used is a query language specifically designed for this task, named SPARQL.

SPARQL

As you have probably guessed, SPARQL is an acronym. It stands for SPARQL Protocol and RDF Query Language. The acronym is defined in terms of itself, a recursive definition. I do believe the people who came up with the idea of semantics liked to have a little fun to lighten up what's otherwise a highly cerebral task.

SPARQL is similar in many respects to SQL. This shouldn't be surprising because both are query languages designed to extract information from databases. Many of the commands and keywords are the same, such as SELECT, WHERE, FROM, DISTINCT, and ORDER BY. Differences arise due to the differences between the data models that the query languages operate on.

SQL operates on relational databases made up of columns representing entity attributes and rows representing entity

instances. SPARQL operates on triple stores, where the subjects are the things being described, the predicates are the names of the properties of those things and the objects are the values of those properties. Subjects and predicates are in the form of IRIs and the objects may either be IRIs or literal values. See Chapter 1 for more information on IRIs.

A SPARQL query retrieves desired information from a triple store by comparing a graph pattern to the triples in the store, looking for triples that match the pattern. Let's consider an example. Suppose the following triple is in the database:

```
<http://dbpedia.org/resource/David_Bowie>
<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/London> .
```

To return all the facts in a triple store about a specified subject, enter a query like the following:

```
SELECT ?p ?o
WHERE { <http://dbpedia.org/resource/David_Bowie> ?p ?o }
```

This query is saying, "Return all the predicate and object pairs in the database where the subject is David Bowie." This query returns the following result:

```
<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/London>
```

In this very small triple store, we can only deduce one thing about David Bowie: that he was born in London.

Note that with RDF, oftentimes you will come across prefixes being used, which help save a lot of typing. Developers can make up their own prefixes, but there are also standard prefixes for commonly used vocabularies. MarkLogic uses the prefix "sem" that stands in for `http://marklogic.com/semantics`. Other common prefixes include the following:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX db: <http://dbpedia/org/resource/>
PREFIX onto: <http://dbpedia.org/ontology/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdfsyntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdfschema#>
```

SPARQL is a rich and powerful query language. There is much more to it than I have space to show here. It is every bit as capable when dealing with triple stores as SQL is when dealing with relational databases.

Aggregates

SPARQL supports many advanced queries, including retrievals that make use of the standard aggregate functions COUNT, SUM, MIN, MAX, and AVG. They act as shown in the following examples:

```
SELECT (COUNT (?company) as ?count_companies)
```

This counts the number of companies in the database and returns the count in the variable named ?count_companies.

This is just part of a real query. Normally you would want to restrict the quantities returned to those that satisfy one or more conditions.

SPARQL supports GROUP BY and HAVING clauses in association with the aggregate functions, as well as FROM and WHERE clauses. These clauses all perform the same functions that they do in SQL. Here's an example:

```
PREFIX demo: <http://mydomain.org>
SELECT ?industry ( COUNT ( ?company ) AS ?count_companies )
    FROM <COMPANIES-GRAPH>
    WHERE { ?company demo:industry ?industry . }
    GROUP BY ?industry
```

This query returns a list of industry sectors along with a count of the number of companies in each sector from the COMPANIES-GRAPH graph.

Inference

The process of inference is that of discovering new facts, based on facts that you already know. In a triple store, *asserted triples,* when combined with *ontology triples,* lead to inferred triples.

Figure 3-1 shows you an example of an inferred triple.

When used in a predicate, "a" is a common abbreviation for isA, and both "a" and "isA" are common abbreviations for the full predicate "rdf:type".

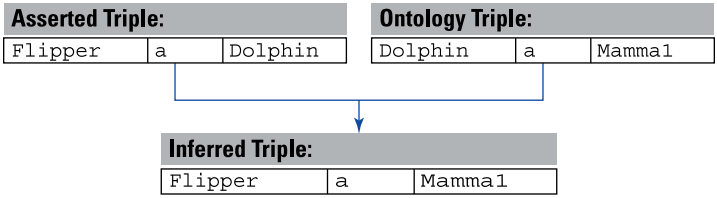

Figure 3-1: An inferred triple.

The Resource Description Framework Schema (RDFS) resides at `www.w3.org/TR/rdf-schema`. The schema provides some meaning to RDF graph data. OWL, the Web Ontology Language, extends RDFS. Triples written in RDFS and OWL contain facts that can give a semantic element to a query. Here's an example of how you can take advantage of the resources of RDFS and OWL:

Suppose you have a triple store containing some people and their occupations, as shown in Figure 3-2.

Example triples in original data:		
David Bowie	a	Singer
Eric Clapton	a	Guitarist
Beethoven	a	Composer

Figure 3-2: Some asserted triples.

Suppose you also have an ontology (Figure 3-3).

Example ontology:		
Composer	rdfs:subClassOf	Musician
Singer	rdfs:subClassOf	Musician
Guitarist	rdfs:subClassOf	Musician

Figure 3-3: Some ontology triples.

Without inference, none of the people who are subjects of the triples in Figure 3-2 would be considered musicians, but with

inference, they all would be. To do inferencing in MarkLogic, you need three things:

- Some asserted triples
- Some ontology triples used to describe your data
- A ruleset such as RDFS or OWL-Horst to define what predicates such as "subClassOf" mean

To illustrate how inferencing works, try to find all the musicians in a triple store supposing that none of the subjects in the store is identified as a musician. Start by asserting some triples:

```
PREFIX dom: <http://mydomain.com/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
INSERT DATA {
    GRAPH <MUSIC> {
        dom:David_Bowie rdf:type "Singer" .
        dom:Eric_Clapton rdf:type "Guitarist" .
        dom:Beethoven rdf:type "Composer" .
    }
}
```

Now that we have some data in the database, let's see what happens when we just run a query without inferencing:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT *
FROM <MUSIC>
WHERE { ?s rdf:type "Musician" }
```

Because you have no subjects in your asserted triples that are of type "Musician", the result of your query is empty.

Now, see what happens when we add some ontology triples to the database, a ruleset, and then rerun a query using inferencing. Here's the ontology triples to add:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdfschema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
INSERT DATA {
    GRAPH <MUSIC> {
        "Composer" rdfs:subClassOf "Musician" .
        "Singer" rdfs:subClassOf "Musician" .
        "Guitarist" rdfs:subClassOf "Musician" .
    }
}
```

Next, you need a ruleset to define the meaning of "subClassOf". MarkLogic includes many rulesets out-of-the-box, including RDFS, RDFS-PLUS, OWL-HORST, and component rulesets for things like "subClassOf". You can use the higher-level rulesets, or just one component — you can even create your own rules and rulesets.

For this example, you specify the rules at query-time. In this case, you specify "rdfs.rules" in your query, which already includes "subClassOf.rules". This approach is very flexible because you only end up using the rules you need for the query you're running at the time.

Also, notice that the SPARQL standard doesn't say anything about how to specify rulesets (or levels of inference). So, in this query example, you'll call SPARQL from JavaScript and specify the ruleset as a parameter to the JavaScript function sem.sparql().

So, now you can write your query and run it:

```
var rdfsStore = sem.rulesetStore(
    "rdfs-plus.rules", sem.store()
    );
var sparqlQuery = " \
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> \
    SELECT * \
    FROM <MUSIC> \
    WHERE { ?s rdf:type 'Musician' . }";
sem.sparql(
    sparqlQuery,
    [],
    [],
    rdfsStore
)
```

The result of that query is

```
[
{"s":"http://mydomain.com/resource/David_Bowie"},
{"s":"http://mydomain.com/resource/Eric_Clapton"},
{"s":"http://mydomain.com/resource/Beethoven"}
]
```

In the query, you specified the ruleset that you needed when you defined rdfsStore. The ruleset rdfs-plus.rules includes a definition of the predicate "subClassOf". The query used this ruleset to perform the inference, which yielded David

Bowie, Eric Clapton, and Beethoven as a match because you defined singers, guitarists, and composers as sub-classes of "Musician" with your ontology triples.

This example uses a type of automatic inferencing called "backward chaining" in which rulesets are applied at query time. This is very flexible because it means you can specify which rulesets and ontologies are applied to each query. An alternative method of inferencing, called "forward chaining," materializes triples when data is first ingested, and is sometimes preferred when your triples and ontology are fairly static.

The ability to infer new information based on existing facts, which may have come from multiple different sources, is a really powerful feature of semantics.

Combination queries

Semantics assumes that your data is in the form of triples and that you will be using SPARQL to query it. A lot of data can indeed be modeled as triples (discussed in the section "Knowing Where Triples Come From"). Other data may be stored as values in a tabular format or as documents. There's value in bringing all of this data into one unified database and being able to search it using what's called a *combination query*.

At present, only one vendor of semantic products, MarkLogic, supports combination queries, but the capability is so valuable that it seems inevitable that others will follow and offer it at some time in the future. To see how this capability may be of value, consider an example from a law enforcement use case.

Imagine that you're the Desk Sergeant at precinct headquarters when a call comes in from an agitated citizen. He relates that a blue van, driving recklessly, nearly hit him and then sped away. In the brief interval after the near collision, he was able to see that the first three letters on the van's license plate were ABC, but he missed the rest. This behavior seems extreme, so you wonder if anyone else has turned in a report on this blue van. You know a few things:

- The subject vehicle is a blue van.
- You know the location of the near collision incident.
- You know the date of the incident.
- You know the first three letters of the license plate.

The relevant data, if it exists, will be in several places in a document, in several datatypes.

- The description of the vehicle ('blue van') will be unstructured full-text.
- The location will be in terms of geospatial coordinates.
- The date will be a value.
- The partial license plate will be in a triple describing the license plate.

A combination query can search for all of the Suspicious Activity Reports (SARs) that contain a match for the "blue van".

```
{
"title": "Suspicious vehicle near airport"
"date": "1987-07023Z",
"type": "observation/surveillance",
"threat": {
    "type": "suspicious activity",
    "category": "suspicious vehicle",
},
"location": [ 37.49705, -122.363319 ],
"description": "A blue van with license plate. . .",
"meta": [
    {"triple": {
        "subject":"IRIID",
        "predicate": "isa",
        "object":license-plate"
        }
    },
    {"triple": {
        "subject": "IRIID",
        "predicate": "value",
        "object": "ABC 123"
        }
    }
]
}
```

Bingo! Another report about the culprit has been identified, and that report contains enough information for us to track down the blue van. Dispatch a squad car. Data has been pulled from a variety of sources and tied together to tell us something we didn't know before. Figure 3-4 gives you an example of a JSON document with four different kinds of data that can all be queried with a single combination query.

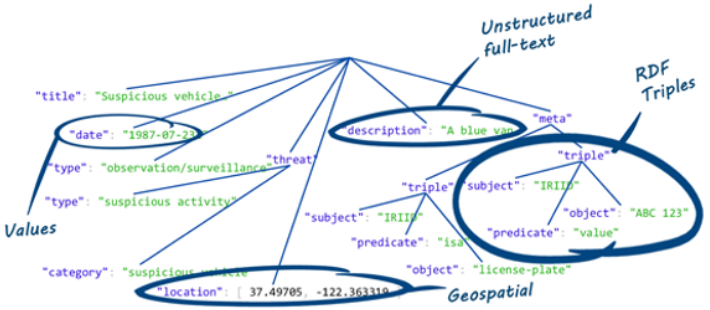

Figure 3-4: A JSON document with four different kinds of data that can all be queried with a single combination query.

Knowing Where Triples Come From

Trillions of triples are already available and listing all of the publically available datasets would be impossible, as more are popping up all the time. Some examples of asserted triples that are publicly available include DBPedia and GeoNames. And, there are also publicly available ontologies, such as Dublin Core, FOAF, and GoodRelations. These publicly available datasets and ontologies provide a resource that you can use right alongside your own in order to answer the questions you are asking. There are even domain specific ontologies that are openly available, such as FIBO for financial services; although sometimes it's best to just create your own ontology and generate your own triples.

DBpedia

DBpedia is a publicly available dataset that has been made under a free license that allows others to use the dataset.

DBpedia pulls its information from Wikipedia. Wikipedia articles are mostly free text, but some structured information is also included in the articles. In the upper right corner of many Wikipedia articles is an "Infobox," that can contain categorization information, images, geospatial coordinates, as well as links to external resources. DBpedia takes this structured information and puts it in a uniform data set that anyone can query.

The resource that DBpedia provides is huge. The data set describes 4.58 million entities. Of which 4.22 million are classified in a consistent ontology. Classifications include people, places, music albums, films, video games, organizations, species, and diseases. There are millions of links to images, external web pages, and even to other RDF data sets. It contains three billion RDF triples, of which 580 million were extracted from the English language edition of Wikipedia. SPARQL queries can extract information that is spread across multiple pages. DBpedia is interlinked with a number of other data sets, enabling applications to infer facts from even more sources than DBpedia itself provides.

GeoNames

GeoNames is a geographical database that stores key information about over 7.5 million places. Information includes place name, latitude, longitude, elevation, population, government type, and postal code. All data is free of charge through a number of different web services. The information is updated daily. Using GeoNames, you can find places next to a given place, via postal code, and also find Wikipedia articles about nearby places. GeoNames also stores "parent features" — so, for example, you can know that "London is in England, and England is in the U.K."

Dublin Core

The Dublin Core is a schema comprising a small set of metadata elements that are useful in describing web resources such as video, images, and web pages, as well as physical items such as books, CDs, DVDs, and works of art. You may use Dublin Core metadata for a variety of purposes, from resource description to combining metadata vocabularies that adhere to different standards, to providing interoperability

for metadata vocabularies in both the Linked Data cloud and the Semantic Web.

Dublin Core derives its name from the fact that it was created at a workshop that took place in Dublin, Ohio, in 1995. The contents of the Dublin Core schema are maintained by the Dublin Core Metadata Initiative (DCMI), which provides an open forum for the development of interoperable online metadata standards.

FOAF

FOAF is an acronym for Friend Of A Friend. It's a machine-readable ontology that describes persons, what they do, and their relationships to other people and to objects. People can add themselves to FOAF as well as how they connect to a social network. It is decentralized and unmoderated, leaving people to maintain their own information and contacts.

FOAF uses RDF and OWL to express descriptive profiles of people. You can use these profiles to find all the people who live in Oregon, or all the people who know Jimmy Wales. When you create a profile for yourself, you can list the people you know. They in turn can list the people they know, enabling the answering of questions such as, "Name all the people that both Allen G. Taylor and Jimmy Wales know." Each profile contains a unique identifier, so there's no confusion over who is who.

GoodRelations

GoodRelations is an ontology that annotates offerings and other aspects of e-commerce on the web. It provides a standard vocabulary for such things as offers to sell a particular product at a given price, offers to provide a service if certain conditions are met, or where a particular product is available from a company with multiple branches.

GoodRelations is flexible enough for you to express just about any details that you want about a product or the business offering it. It operates under a Creative Commons Attribution 3.0 license, which means you can use it for free as long as you attribute the work to its source.

Creating Triples and Managing Ontologies

Database products that store and operate on triples must first ingest those triples. If the triples already exist, this can be a fairly straightforward operation. If not, there are countless ways in which you can create your own triples. The easiest way would be to hand-code whatever triples you want. But, given that's not very scalable, a number of better ways have evolved.

If you have relational data that you want in a triples format, you could write a small Java program to parse the data into triples. You could also use R2RML, a language developed specifically for expressing customized mappings from relational databases to RDF datasets (more information on R2RML is available at `http://www.w3.org/TR/r2rml/`).There are also scripts that have been written to create triples from XML documents or from strings.

If you want to employ a tool to make things easier, there are a variety of options that can help you with creating and managing ontologies, extracting triples from unstructured text (called entity extraction), and much more. Some tools we discuss below include

- Smartlogic (`http://www.smartlogic.com`)
- Protégé (`http://protege.stanford.edu`)
- Temis (`http://www.temis.com`)
- Pool Party (`http://www.poolparty.biz`)

These are just a few examples, but the truth is that there are dozens of approaches to creating and managing your own triples, and choosing the right approach depends on the size and type of data you have, and what application you are building.

The tools listed in this section are just a few examples of the many that can be used in conjunction with a database such as MarkLogic, where the triples would be stored and managed.

Smartlogic

Smartlogic is a software company whose main product, Semaphore, classifies database content using taxonomies and ontologies. It automatically generates metadata that represents each classification decision. With metadata, you can create workflow processes and handle sensitive data in special ways. Semaphore uses semantics, text analytics, and visualization technologies to perform five primary functions:

- Ontology and taxonomy management
- Automatic classification of unstructured data
- Text analysis
- Metadata management
- Content visualization

Protégé

Protégé is a free, open-source ontology editor that originated at Stanford University. It can serve as a framework for building intelligent systems. With Protégé you can build both simple and complex ontology-based applications. Developers can build a wide range of intelligent systems by integrating the output of Protégé with rule systems or other problem solvers. For example, in Chapter 2 we mention an application built using the Object-Based Intelligence (OBI) framework, which used Protégé to create an ontology.

Temis

Temis markets the Luxid content enrichment platform, which structures, manages, and exploits unstructured content. It works by identifying and extracting targeted information to semantically enrich it with domain-specific metadata. It can work with entities of general interest such as Companies, People, Locations, or on those that are more domain-specific, such as Proteins or Genes in biology. Major application areas are content publishing and enterprise content management.

PoolParty

PoolParty is another semantic technology platform that serves as a thesaurus and ontology management system, and can also do text mining and entity extraction. The PoolParty API makes it easy to integrate their family of tools with enterprise search, triple stores, and open linked data. It is also designed to run easily in the cloud. PoolParty is a product of the Semantic Web Company, based in Vienna, Austria.

Chapter 4

Comparing Technologies

In This Chapter

- Comparing RDF triple stores and relational databases
- Looking at integrated NoSQL databases versus stand-alone triple stores
- Differentiating RDF triple stores from graph databases
- Relating semantics and linked data

Whereas in the past, relational database technology was just about the only game in town when it came to storing business data, the situation has changed radically in the past several years. Now there are more options, and as a result, it's now possible to handle more kinds of data and to do more with the data you have. This chapter talks about the various data storage technologies that are available today and how they stack up against each other.

RDF Triple Stores versus Relational Databases

Relational databases have been the dominant tool for storing business data for the past 35 years — they are fast, reliable, and secure. What more could a person want?

Unfortunately, relational databases come with trade-offs. They require that extensive time be spent on modeling data, transforming data to fit the model, and the inability to make changes later. You pretty much need to know exactly what questions you want to ask and how the data should be modeled to answer your questions with adequate performance.

After you have loaded the database with data, it's very difficult to change the structure. Sometimes it's impossible, and you must toss out what you have and start afresh. The relational data modeling exercise can take years for large projects and cost millions of dollars. This increasingly is a tough trade — today's data doesn't fit neatly into rows and columns like you want it to, and it changes fast. With rich relationships and hierarchies, the problem of sparse data, and the enormous volumes that big data brings, it's really hard to preserve a relational model.

Additional problems with relational databases include the following:

- They work very well for data that can be arranged in rows and columns like the rows and columns of a spreadsheet, but they don't work well at all for data that doesn't fit into that paradigm or has a complex structure — such as documents and graphs (think of things like derivative trade documents, healthcare records, articles, blog posts, repair manuals, social media profiles, and so on).
- Scaling is an issue when databases get really large because relational databases scale up vertically rather than scaling out horizontally.
- Relational databases have to have pre-defined schema for anything you might want to store, which leads to the problem of sparse data. If there's not a column for something, it's ignored or ends up in a "varchar" column.
- Human readability is a challenge that's familiar to anyone who has seen columns in a relational database with names like "fname" and "lname".
- Queries with a lot of joins and a high volume of data can run slow or require de-normalization or data duplication.

Semantics offers flexibility that was lacking with relational databases. Semantics offers the ability to preserve relationships, link relationships, and annotate data later. When you combine the semantic data model by using RDF with a NoSQL document model that stores data as documents, you get the ultimate in schema-less data models. It makes it easy to create, easy to combine, easy to share, and when used in combination with documents and data, it can do all the things a relational database struggles with.

RDF triple stores are architecturally entirely different. The subjects and objects of triples are stored as graph nodes, and the predicates that relate them are stored as graph edges. No space is wasted on non-existent data. One way flexibility comes into the picture is through the fact that any number of edges can connect to a single node, and a graph made up of interconnected triples can grow in any way that the data dictates.

There's no need to try to shoehorn the data into a row and column organization that may be inappropriate for many of the use cases that businesses are concerned about today. And, some triple stores don't give up the ability to run transactions on your data — you can still support transactions with high data integrity requirements, known as ACID compliance.

Integrated NoSQL Databases versus Stand-Alone Triple Stores

A triple store is a database that stores facts in the form of triples and retrieves desired information with some query language, such as SPARQL 1.1. On the other hand, an *integrated* NoSQL database includes a triple store, plus something else. Sometimes this is referred to as a *multi-model* database. As marvelous as triples are for expressing a relationship between a subject and an object, you may want to know some things that are difficult to express in that format. For example, suppose you have a triple store that contains the triple "John livesIn London". Now suppose you want to say that the source of the triple was the BBC, that you received it ten days ago, and that you are 80 percent confident that it's correct. Trying to express all this with triples would be very difficult, not to mention frustrating.

The fact is, there are sometimes better models for certain types of data. In many cases, storing data as documents in a document-oriented NoSQL database makes more sense than trying to force data into a semantic model. The important thing is making sure the database and data model is the right fit for your data, while also avoiding having to manage a polyglot of various technologies.

Although the value of having an integrated, or multi-model, database seems evident, only a few products have such capabilities at present. With them, you can choose to represent as much or as little as you want in each model. You could have a customer record in JSON or XML and also link the customer ID in the document to a credit rating in a triple so it can be easily queried with other credit ratings. Another way to mix the two data models is by embedding triples in JSON or XML documents.

RDF Triple Stores versus Graph Databases

Commonalities and differences exist between RDF triple stores and graph databases (more technically called "property graph databases"). RDF triple stores differ from graph databases in that (surprise!) they're based on triples, while graph databases are based on nodes and edges.

Table 4-1 shows you a comparison of the two.

Table 4-1 A Comparison of Triple Stores and Graph Databases

Triple Stores	*Graph Databases*
Uses W3C specifications (RDF, SPARQL)	Use product-specific specifications (for example, Cypher query language)
Schema-less: It's all just triples	Schema-based: Need to define types, properties, values up front
Optimized for aggregate queries	Optimized for graph traversal and "shortest path" queries
Optimized for arbitrary URLs	Long URLs should be avoided
Built to link disparate data	Sharing data is challenging
Built for rules and complex reasoning	Requires procedural Java code
Built for inferencing	Not designed for inferencing
Easy to add metadata to nodes	Difficult to add metadata to nodes

Semantics versus Linked Data and Other Similar Technologies

Clearly, semantics, with its triples, is a kind of linked data, and the two concepts are related. However, when you see the capitalized words "Linked Data" they have a very specific meaning that doesn't necessarily imply semantics. Linked Data is structured data that's interlinked in some way. It may be used in semantic queries. Or not.

Semantics

In the context of databases, semantics deals with modeling data in a flexible way that still provides meaning and context. With the data modeled as triples in a graph, new facts can be inferred through a search process that follows links and creates new relationships based on what it finds. By integrating semantic data in a triple store with textual data in documents, combination queries can discover relationships that may have been undiscoverable with traditional tools.

Linked Data

Linked Data uses the standard web technologies, such as the HTTP protocol, the RDF framework, and the URI (or IRI) identifiers. However, it uses these things in a different way. Rather than using them to point to a web address, it uses them to identify other types of resources.

The Linked Data terminology was coined by Tim Berners-Lee, the inventor of the World Wide Web and Director of W3C. He stated four principles for the use of Linked Data:

- Use URIs to denote things.
- Use HTTP URIs so that the things denoted can be looked up ("dereferenced").
- Provide useful information about the thing being dereferenced, using RDF and SPARQL.
- Include links to other related things, using their URIs, when publishing data on the web.

Open data

The concept of *open data* is that certain data should be free of conventional restrictions that limit or control its use. In this sense, open data is to data what open source is to computer source code. The open data idea is often applied to material other than text, such as maps, genomes, connectomes, chemical compounds, mathematical and scientific formulae, and medical data. There are several government sponsored collections of open data, which can be found, for instance, at Data.gov and Data.gov.uk.

Data.gov has databases in many areas of interest. In the agricultural area it has for example, a food desert database. This database shows localities in the United States that are classified as "food deserts." A food desert is an area where a substantial fraction of the residents are classified as low income, live more than half a mile from a supermarket, and do not own a car. City planners can take this information into account when deciding on the granting of building permits. Citizens who are considering moving to an area, may decide to avoid food deserts, particularly if they do not fancy carrying heavy groceries long distances.

Linked open data

I'm sure you have already figured out what linked open data is, assuming you have read the two previous sections. It is linked data (or Linked Data) that is free for anyone to use, reuse, or redistribute, subject only to attribution of the source. DBpedia is an example of a repository of linked open data.

The Semantic Web

The Semantic Web is an extension of the World Wide Web that follows standards developed by the World Wide Web Consortium (W3C). The Semantic Web provides a common framework (RDF), facilitating the sharing and reuse of data across traditional boundaries. It extends the web's hyperlinked collection of human-readable web pages with machine-readable metadata that describes the pages and how they relate to each other. As a result, automated agents can

perform more tasks for users. Machines can now perform the mundane tasks that involve finding, collating, and acting on data on the web.

Artificial intelligence

You might be wondering what connection semantics may have to artificial intelligence, known by its initials, AI. After all, semantics uncovers meaning embedded in data, and that's a sign of intelligence, isn't it?

Well, no. Computer scientists have been trying to create artificial intelligence for half a century. They have had some limited success, such as IBM's Deep Blue supercomputer, which beat the World Chess Champion, Gary Kasparov, in a match a couple of decades ago. Apple's Siri and Microsoft's Cortana are examples of very rudimentary AI, doing natural language processing (NLP) to understand what you are saying. Semantics, although developed entirely separately from AI projects, can be used for natural language processing. It's also probable that at some point in the future AI investigators will start taking advantage of the powerful inferencing capabilities of semantics. By the time we do achieve AI, though, we probably won't call it AI — that goal keeps getting pushed out and will always remain elusive in the world of computing.

Semantics has developed entirely separately from those projects. However, it's probable that at some time in the future the AI investigators will start taking advantage of the powerful inferencing capabilities of semantics.

Chapter 5

Ten Things to Watch Out For with Semantics

In This Chapter

- Knowing when, and when not, to use semantics
- Making sure your security is in check
- Achieving high performance with semantics

This chapter summarizes, in ten (okay, nine) easy to digest nuggets of wisdom, the most important things to know about semantics.

Recognizing Opportunities to Use Semantics

There are classes of applications for which semantics is a good technology to use in terms of speed and ease of development as well as performance of the resulting application. Some tasks aren't a great choice for semantics. It's important to know the requirements of a job as well as the strengths and weaknesses of all architectures that you may use before committing to a semantics implementation or on what works according to some other model.

When you get to work on building your next big application, you may want to use semantics if you want to

- Model atomic facts about people, places, and things
- Model complex relationships

- Share your data using a common standard
- Discover "hidden" facts in your data
- Visualize your data as a graph
- Build a more intelligent search application
- Extract meaning from unstructured data
- Classify large amounts of data
- Work with open linked data
- Reconcile and integrate disparate data
- Provide context for a specific domain of knowledge
- Automate publishing of facts

Recognizing When Not to Use Semantics

The question of when to use or not use semantics is often driven prematurely by how the data is currently being stored. If data is already stored in rows and columns in a relational database, then the opportunity cost of reformatting that data is often perceived as being too high.

The truth is that RDF works quite well for storing data that could be stored in a relational database, or for data that is stored in a document database. The real answer to "when not to use semantics," assuming you have a choice from the start, should focus more on how much flexibility and context you want with your data. If you don't need flexibility and you don't care about the relationships in the data, then you may not need semantics.

Also, if you think you have a need for RDF, but don't see how your RDF would integrate into downstream processes or with your other data stored in other databases, then you may just be creating another silo. This isn't an issue with integrated databases such as MarkLogic that can store RDF alongside other data models, but does come up with specialized triple stores that were created for a single purpose.

Some of your data may fit more naturally into JSON or XML documents. For example, if you have some pieces of data that are always stored, queried, and reported together — such as a customer record of name, address, and phone number — it doesn't make sense to break that record apart into triples and reassemble it every time you want it. In this case, store the customer record as JSON or XML, and link that document to triples and to other documents.

Standardizing with SPARQL 1.1

Just as SQL is the industry standard language for creating, maintaining, and querying relational databases, SPARQL 1.1 fills the same role for databases that follow the RDF data model. SPARQL started out as merely a query language, but the 1.1 version has rounded out the feature set so you can now use it to insert and delete triples and manage graphs as well.

Before SPARQL became a W3C standard, many different NoSQL databases had their own home-grown query languages, but as soon as SPARQL received the blessing of W3C, more and more database vendors migrated to SPARQL. With the release of the SPARQL 1.1 specification in 2013, SPARQL has for all practical purposes become a mandatory feature of any RDF database. In contrast, there's no standard query language for graph databases. For portability considerations and the long term support of a database, standards are definitely something that you want.

Exposing SPARQL Endpoints to the World

A SPARQL endpoint is a RESTful service that enables a dataset to be queried with SPARQL. The endpoint is identified with a URL and it accepts SPARQL queries and return results to the issuer of the query. If you expose that endpoint to the world by making it accessible via the web and don't secure it with a username and password, your data is now available to anyone who chances to ask for it, including your competitors and enemies. And, a novice user could even accidentally write a SPARQL query that "joins" everything and brings down an

entire server. It is for this reason that some "open data" sites no longer have SPARQL endpoints open to the public.

Think carefully about exactly what you are making available before exposing a SPARQL endpoint over the web.

Securing Your Triples

The security of data is a major concern of enterprise level organizations, not to mention smaller outfits whose primary asset is the data they have that their competitors don't. In the case of triple stores, inadequate security is often a deal breaker. Stand-alone triple stores typically don't have robust security and haven't gone through any kind of certification process. If the security of your data is important to you, take a careful look at what's being offered by your triple store vendor.

One solution to this is to use a system where the triple store is integrated as part of an enterprise database system and benefits from the security regime in place for it. As an example, MarkLogic has certified security using role-based access control and can secure individual triples and collections of triples just it secures individual documents and collections of documents. Two users making the same query would see two different results, depending on the security level of their individual logins. Most specialized triple stores don't have this high level of control and haven't achieved common criteria certification, so you have to be careful when considering mission-critical applications.

Indexing

The whole point of indexing your data is to make search fast. Different triple stores have different ways of indexing their data. Depending on your application, you may find some of these indexing schemes perform well for you and others don't. This is definitely something to watch out for. If you have a large database (and you are probably not reading this book if that isn't the case), make sure that the indexing scheme in use in the product you're considering is one that gives you

fast responses to your queries even as your triple store grows and that it doesn't limit the size of your triple store.

Achieving High Performance

One aspect of achieving high performance is to maintain appropriate indexes for the kinds of tasks your application usually performs. Another has to do with the architecture of the particular system you're using. Many triple stores have everything in memory. This is great as long as your memory is big enough to hold everything, but it puts you in serious trouble if your store grows beyond what the memory can hold. This can also be an expensive proposition, because you will have to buy a *lot* of memory.

Some stores, however, such as MarkLogic, have a triple cache, and triples are not memory-mapped. This is a much more scalable architecture, so you don't have to worry about outgrowing what your hardware can support.

Scaling Big

Big Data is the natural application area for semantics. It was the arrival of truly huge data sets, thanks to sensors and other instruments generating massive streams of data, that really ignited the push to NoSQL databases in general and semantics in particular. A number of companies have product offerings to address this market, but there can be a major difference in cost that doesn't necessarily track with capability. If you can get your job done with commodity hardware that's inexpensive to buy and to maintain, there's little incentive to look at more expensive alternatives.

In addition to hardware, it's also important to understand how the solution shards data across a cluster and how query performance changes as data sets grow. We've already mentioned that few graph databases scale because it's very difficult to shard graph data, but it is also difficult to scale triple stores. If you want to grow your triple store to trillions of triples, it's important to ensure you have an enterprise-class triple store that is proven in production to hold up under pressure.

Integrating Semantics

You get a big benefit by using an integrated, multi-model approach in which you store and manage your triples right alongside your other data — all in the same database. Until recently, triples were largely seen as an esoteric way for academic types to model data. Triples usually ended up sitting all by themselves in stand-alone triple stores without much real use, despite their intrinsic value.

But, things have changed. RDF is now going mainstream. Today, major organizations are building powerful applications by using an integrated approach to semantics in which RDF data is stored right alongside other data. This integrated approach is becoming the standard way to utilize semantics. It provides flexibility to use the right data model and right query language at the right time. And it provides the flexibility to change the approach on the fly.

Although few vendors currently offer it, getting the full value out of semantics requires an integrated system that's able to seamlessly operate on documents, structured data, and triples together. If data in all three forms is part of your concern, such an integrated data model is the best way to maximize the value you receive from your assets. It will return results to you that you couldn't get from a system that focused only on data in one form or another. Look for an integrated solution — one where all the elements are tuned to work together.

Notes

Notes