2

Modelling

Learning outcomes of this chapter

- **Putting linked data in a larger context**
- **Getting out of a hype-driven view on technology**
- **Understanding the importance of data modelling**
- **Making sense of data models and their serialization formats**

1 Introduction

Metadata managers and the software they use often seem to have a striking resemblance with couples stuck in an unhappy relationship. During coffee breaks at conferences and workshops on metadata within the library and information science domain, it will not take you long to spot a circle of people engaged in what seems to be some form of group therapy. Do not be afraid. Go ahead and stand a bit closer. You will probably overhear typical phrases such as 'We have been struggling to create new metadata fields for years!' or 'My XML export is terrible!' Confronted with these laments, the group members will nod understandingly and express their sympathy. Complaining about one's software is a popular point of discussion across the globe when collection holders come together to discuss metadata. Ironically, just like old couples who think twice about divorce due to the important emotional and economic consequences, metadata managers often persist for years in the abusive relationship with their software. They usually prefer not to move over to another software solution.

How different the ambiance in the digital humanities! Instead of complaining about the difficulties encountered with their database, people active in the digital humanities often are very proud about the information system they built to manage a specific type of resource or collection. Susan Hockey even coined the expression '"Me and my database" papers'. Anyone who has already attended a DH conference is familiar with the phenomenon: a researcher who presents, in tedious detail, how a database was developed to accommodate every peculiar

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

feature of a collection. These speakers tend to be very proud of the database they constructed and radiate love and passion for it.

Why do these two communities have such a different approach to the software they use to manage cultural heritage resources and their metadata? Why do collection holders constantly whine about their database, whereas digital humanists express their satisfaction and even brag about the happiness they found with their database?

These differences relate to the extent to which the *model* used to represent an object and its metadata is deemed adequate or not. When we want to make resources and their metadata available in a structured manner on the web, we first need to decide what characteristics of theirs are the most important to be represented. By doing so, we make an abstraction of the reality through the development of a model.

In the cultural heritage context we mentioned, institutions are forced to work with off-the-shelf software, since the development of a custom-built collection management system is simply not economically feasible. The drawback of working with existing software is that institutions often find themselves limited in how they can describe their objects. Vendors have a commercial incentive to develop generic software that can be sold to as many institutions as possible. This implies that collection management software already prescribes a certain explicit *worldview*, through the use of a pre-established model. It is therefore not always possible to accommodate the specific requirements of an institution and its collections, leading to frustration amongst collection holders.

In contrast, the DH community uses databases for limited and specific research projects, as they tend to focus on the documentation and publication of one specific type of resource or collection. Within these limited research projects with a precise scope, the requirements tend to be so specific that it is not possible to use off-the-shelf software. In this type of context, relational database management software (RDMS) is often used to implement a tailored model. The drawback of meeting all the precise requirements of such a project are the relatively high development costs and the difficulty to maintain the application over time. Investments are made in projects that very often cannot be re-used.

1.1 Deciding where to put the semantics

What does this have to do with linked data? The examples above demonstrate that both the use of a generic, standardized model and of a highly customized, specific model come at a cost. The tremendous amount of effort the LIS community has put into metadata standardization reflects how we have been trying to find a sweet spot between the two approaches. As it will be demonstrated through practical examples, the evolution from an unstructured

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

narrative to a highly structured representation of metadata requires the development of schemas in order to make the metadata interoperable. By slicing up unstructured descriptive narratives into well structured fields, we need to render the meaning of the different fields (also called attributes) explicit by documenting them in a *schema*. By structuring and atomizing metadata fields we make them more machine-interoperable, but we also become more and more reliant on the schemas when needing to interpret our own or someone else's metadata. It is precisely in this context that linked data need to be understood. Through the adoption of a radically simple data model, abstraction can be made of the traditional XML and database schemas we had to use in the past to interpret and re-use data.

1.2 Getting away from a hype-driven view of technology

The adoption of a new technology is often illustrated in the form of the famous Gartner hype cycle (Lynden and Fenn, 2003). The graphical representation of the rise and decline of the popularity of a new technology draws attention to the exaggerated expectations which often accompany its introduction. After the so-called *peak of inflated expectations*, a technology tends to lose most of its appeal on the market a couple of years after its introduction. It is only after an extensive period that the technology reaches a stable level of adoption, based on its genuine added value in a production environment. One of the goals of this book is to teach you how to step away from a hype-driven view on technology by helping you understand not only the exact added value of linked data, but also its weak points.

Where should we situate linked data in this cycle? The recent enthusiasm to connect heterogeneous resources and to draw in new information from external knowledge sources perhaps recalls for some the unbounded enthusiasm the cultural heritage sector had for the eXtensible Markup Language (XML) around 2000 and a couple of years later for web 2.0. In hindsight, we can now safely say that both approaches have been (and continue to be) fundamentally important for how we create and manage our metadata. However, we should also acknowledge that neither XML nor the social web resolved all of the fundamental problems underlying how we can connect resources from various collections.

Despite a major overhaul of the general technological framework, illustrated by other developments, such as the maturing of open-source collection management systems and cloud-based hosting, for example, we are still very much facing the same problems the cultural heritage community was discussing almost five decades ago. For anyone working on the topic of digital cultural heritage, it is a humbling experience to read about the discussions that were taking place in the 1960s and 70s. In parallel with the creation of the Computer

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

Museum Network in 1967, a project was launched to create a common collection management database that would be used by all participants of the consortium. Numerous other initiatives have since been based on the same fallacy: if we all use the same tool, our metadata will become interoperable. Again and again, projects have demonstrated that even if people and institutions are using the same tools and standards, they implement them in different ways to accommodate the specific nature of their collections.

Are linked data here to break this vicious circle, or are we again confronted with an overhyped technology? Before we answer that question, we need to moderate the inflated expectations surrounding linked data. Practitioners trying to get to grips with linked data principles are frequently frustrated when confronted with the output of large-scale IT research projects. Huge volumes of metadata and controlled vocabularies have been converted over recent years into Resource Description Framework (RDF), producing billions of RDF statements. Unfortunately, these so-called *triple stores* only unlock their value through the use of a complex query language called SPARQL Protocol and RDF Query Language (SPARQL). The purely technology-driven nature of many linked data projects is leaving a bitter aftertaste amongst practitioners, who feel they need a PhD in semantic web technologies in order to take advantage of linked data.

1.3 The world's shortest introduction to data modelling

Let us therefore, in this chapter, step away from the merely hype-driven view of linked data by choosing a more conceptual and historical approach. In order to grasp the potential but also the limits of linked data, we need a better understanding of the different data models which have been used over recent decades to manage metadata. The advantages of RDF, the data model underlying the linked data vision, can only be fully understood in the context of previous data models. At the end of this chapter you will understand that the different data models presented do not supplant one another, but continue to co-exist. The overview of the different models should make it clear that relational databases are here to stay, and will not be disposed of in favour of triple stores. Technology vendors and IT researchers have a tendency to overemphasize the role a new technology has to play. At the height of the popularity of XML, one sometimes got the impression that the back-end of any type of information system would become XML-based. A decade later, XML is criticized more often than not, and new serialization formats such as JSON are often preferred. This chapter will provide the world's shortest introduction to IT fashion, in order to help you see the wood for the trees.

We will specifically focus within the overview offered in this chapter on the management of structured data, but be aware that the traditional barriers between structured and unstructured data are becoming increasingly blurred.

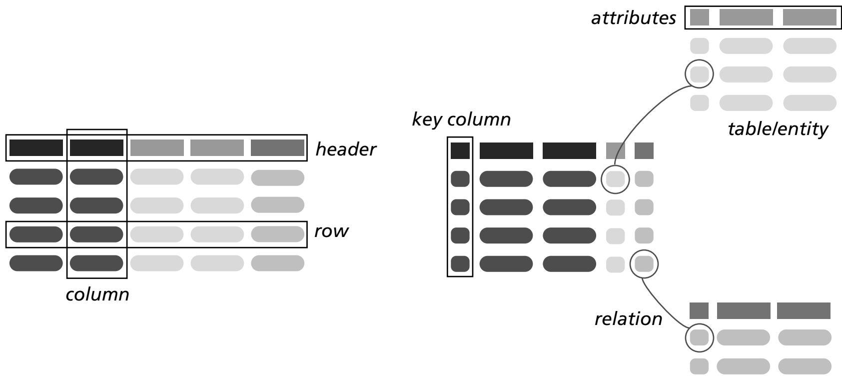
For more Facet books and to order online visit: http://www.facetpublishing.co.uk

For decades, different communities have been working independently from one another on both topics. Database engineers focus on the optimization of the management of structured data, whereas computational linguists develop methods and tools to manage unstructured natural language in an automated manner. The different traditions and views between the two communities can be illustrated by analysing how both communities make use of XML. Computer engineers see XML as a hierarchical tree in which structured data can be encoded in order to facilitate the communication of data between machines. On the other hand, computational linguists and digital humanists look at XML as a method to insert small pieces of structure into an otherwise unstructured textual document. Indicating where exactly in a full text the names of places or people can be found allows scholars to automate to a certain extent the analysis of an unstructured corpus. We will discuss XML in more detail in the section on meta-markup languages later in the chapter. The traditional distinction between structured and unstructured data is particularly problematic in the context of metadata. For example, within a highly structured metadata record a descriptive field might occur which contains a narrative of several pages of unstructured full text. Should this metadata record be considered structured or unstructured?

The chapter will start with the most intuitive model for structuring data, which is *tabular formats*. Due to the limitations of this approach, the *relational model* was developed in the 1970s, remaining until today the standard to represent and manage complex data. As will be explained over the next sections, the appearance of the web towards the end of the 1990s catalysed the need for *data portability*. Sharing data between different databases is a very tedious process, for reasons which will be explained below. In order to facilitate the exchange of structured data on the web, meta-markup languages, and XML in particular, have been used from 2000 onwards. XML proposes a standardized syntax for the automated exchange of structured data, but the actual use and interpretation of the data can still be troublesome. The meaning of the elements and attributes of the XML files need to be defined in a schema. The interpretation of the schema remains a barrier for an automated consumption of data across information systems on the web. It is exactly here where RDF comes in. By adopting a data model which embodies the meaning of the data in its most essential and stripped down form, there no longer is a need for an outside schema to interpret and re-use the data.

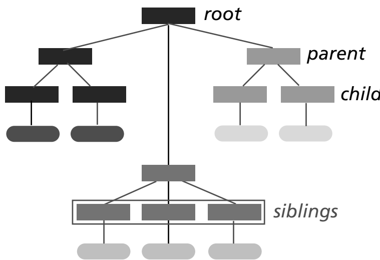
Figure 2.1 compares the different models from a high-level perspective. You could consider this figure as a synthetic overview of Chapter 2. We are conscious that we are covering a lot of ground with this chapter. At times, it might be challenging to understand the interaction and links between the four different data models to be discussed. In order to help you put the individual sections of this chapter into a bigger perspective, Figure 2.1 highlights in an abstract manner the features of each data model. Even though each model has its own properties,

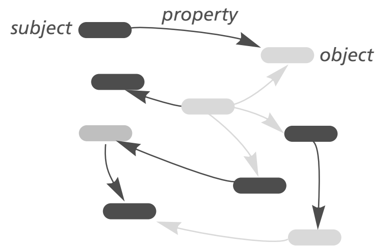
For more Facet books and to order online visit: http://www.facetpublishing.co.uk



Tabular data
Each data item is structured as a line of field values. Fields are the same for all items; a header line can indicate their name.

Relational model
Data are structured as tables, each of which has its own set of attributes. Records in one table can relate to others by referencing their key column.





Meta-markup languages
XML documents have a hierarchical structure, which gives them a tree-like appearance. Each element can have one or more children; there is exactly one root element.

RDF
Each fact about a data item is expressed as a triple, which connects a subject to an object through a precise relationship. This leads to graph-structured data that can take any shape.

Figure 2.1 Schematic comparison of the four major data models

similarities have been highlighted insofar as possible. For example, rounded shapes represent individual data values; rectangular shapes indicate model-specific ways to add structure (with the exception of RDF, where arrows are used). Different shades indicate data values that semantically belong together, indicating how different models treat them.

1.4 Every advantage has its disadvantage (and vice versa)

Be aware that this chapter explicitly does not represent the different models as a linear succession of increasingly high-performing solutions to manage structured

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

data. In other words: please do not interpret the following sections of this chapter as a story of how we have gone from an inadequate approach towards the perfect solution. As with many things in life, advantages offered by a data model often imply disadvantages (and vice versa). For example, the schema-neutral feature of RDF comes at a big cost. Whether a data model (and the methods, technologies and tools it comes with) is suited for you entirely depends on the context of the problem you want to solve. To make things as clear as possible, every model will be illustrated with the help of small examples of metadata in relation to the work of Pablo Picasso.[1]

1.5 Data models and their serialization formats

Before we proceed to the overview of data modelling, it is important to clearly distinguish each model conceptually from the formats that have been developed to *serialize* the data model. The serialization process converts data structures into a *format*. The format allows the translation of the information you are representing into a stream of bits that can be manipulated by software, communicated over a network, etc. The format allows a conversion from the bit level back into the original data.

The difference between a model and a serialization can be compared to a dish and its recipe. The in-memory model is the thing itself and is accessible for manipulation – or is immediately ready to be eaten, like a prepared dish. A serialization contains all necessary elements to construct the in-memory model, just as a recipe contains all the information you need to prepare the dish. Table 2.1 gives an overview of the different data models and their serialization formats which will be discussed over the next sections.

Table 2.1 Data models and their serialization formats

data model	**serialization formats**
tabular data	CSV, TSV
relational model	*proprietary binary files*
meta-markup languages	XML, SGML
RDF	Turtle, N-Triples, RDF-XML

2 Tabular data

Suppose you were given a collection of resources, such as photographs, books or DVDs, and were asked to describe the collection. What would be the most intuitive and natural thing to do? Chances are high that you would take a sheet of paper, or create a spreadsheet on your computer, and create columns in which you will aggregate the most important metadata of the resources, such as title, creator, date, etc.

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

2.1 Model

Conceptually, the world view you create with tabular data is comprised of columns and rows. The intersection of a column with a row gives meaning to the data contained in the particular cell. Figure 2.1 illustrates this data model on an abstract level. There is only one modelling dimension, consisting of the fields in the first header row. Each row contains data from different semantic entities, which we can also refer to as *records*. This is why tabular data are often referred to as *flat files*. Coming back to our concrete example, it is an intuitive act to use this model and to draw up a list as presented in Table 2.2, illustrating how you might develop a tabular overview of your resources.

Table 2.2 Example of metadata encoded as tabular data

title	creator	date	collection
Guernica	Pablo Picasso	1937	Museo Reina Sofia
First Communion	Picasso	1895	Museo Picasso
Puppy	Koons, Jeff	1992	Guggenheim
...	...	...	...

Over centuries, catalogues and indexes were encoded in this tabulated form. The list, as most people would call tabular data, can probably be considered as the oldest information technology. Drawing up lists organized in columns is still often the first step taken when brainstorming and developing ideas about what metadata elements should be used to document a resource. Why is this such an intuitive data model? Tabular data offer the big advantage that they are almost self-explanatory. When reading a catalogue or an index in this format, you have a natural tendency to read in a horizontal manner by focusing on one line of the catalogue and reading from left to right the information gathered in the different 'boxes'. This allows you to get an immediate overview of all the different metadata elements (in our table: *title*, *creator* and *date*) concerning one specific object. Semiologists or linguists would refer to the importance of the *syntagmatic* relations. Through the combination of different elements, meaning is created in the sense that we understand what an object is, when it was created and by whom. A vertical reading, gazing up and down the columns, allows you to get a sense of the different values of one specific metadata element. On this level, the so-called *paradigmatic* relations operate. These relations cluster members of the same category.

The difference and interaction between syntagmatic and paradigmatic relations might seem like a pedantic academy side note. Keep in mind that they play an important role in understanding the difference between the use of unstructured descriptions, which we can refer to as narratives, and structured metadata fields, which have been sliced up to make them more machine-processable. Lev Manovich drew attention to the fundamental difference between these two forms of presenting information:

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

> As a cultural form, a database represents the world as a list of items and it refuses to order this list. In contrast, a narrative creates a cause-and-effect trajectory of seemingly unordered items. Competing for the same territory of human culture, each claims an exclusive right to make meaning out of the world. The database (the paradigm) is given material existence, while the narrative (the syntagm) is dematerialized. Paradigm is privileged, syntagm is downplayed. Paradigm is real, syntagm is virtual.
>
> Manovich, 2001, 231

Traditionally, people have privileged the form of narrative when communicating information, but the massive presence of database-driven applications on the web is reversing the situation. This evolution is very much embodied in how our metadata practices have evolved. From the beginning of the 20th century, our cultural heritage institutions have started to decompose the lengthy narrative descriptions drawn up by curators and transferred them to card catalogues and database records. This evolution drastically helped to facilitate search and retrieval, but actually making sense of a complex object is still based on the unstructured description. Manovich goes too far when presenting the two forms as exclusive and competing. It is the introduction of database-driven websites that made the advent of web 2.0 applications possible. As Chapter 5 will demonstrate, user comments can offer a valuable enrichment of the limited metadata an institution can provide, whereas named-entity recognition (NER) can currently be applied to facilitate complex search and retrieval procedures based on unstructured full text in natural language.

2.2 Serialization

The most popular serialization formats of tabular data are comma-separated values (CSV) and tab-separated values (TSV). The only, but important, difference between these two formats are the characters, appropriately called delimiters, used to indicate the separation between values. As their name indicates, CSV files use a comma as a delimiter, and TSV tabs. Please note that any type of character can be used as a delimiter. The CSV data from Table 2.2 are separated by a *comma* and rows are ended with a *line break* as follows:

```
title,creator,date,collection
Guernica,Pablo Picasso,1937,Museo Reina Sofia
First Communion,Picasso,1895,Museo Picasso
Puppy,"Koons, Jeff",1992,Guggenheim
```

The TSV version would look exactly the same, but the commas would be replaced by tab characters. And strictly speaking, the quotes around `Koons, Jeff` would

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

not be necessary because the comma has no special meaning. If, however, a value needs to contain an actual tab character, quotes would be necessary.

2.3 Search and retrieval

What are the implications of this data model for search and retrieval of metadata? A quick look at the metadata in Table 2.2 gives an overview of the limitations of the tabular file approach. For example, the name of the creator is expressed in different manners ('Pablo Picasso' and 'Picasso'), as we encode the same reality every time when describing a new object this artist made. For a human being, it is straightforward to map these two different representations to the same reality. You have probably also noted the presence of "Koons, Jeff", in which quotes are used to protect the comma separating the family and the first name.

When performing a full-text search on a string of characters, an algorithm will have more problems to deliver good search results. Let us therefore suppose that you want to update your metadata and encode the name of the creator in a uniform manner. Now imagine you do not have three records (as it is the case in our example) but a couple of hundred thousand . . . Managing your metadata in a tabular list would imply that you would need to go through all these records to see where one of the different spellings of the creator's name appears and update it if needed to the preferred spelling. This working method is bound to introduce *inconsistencies* in your metadata. On a computational level, search and retrieval is very inefficient with this approach, as again the totality of your metadata records have to be checked to see whether they contain a specific value. Neither do tabular files offer the right tools to impose rules on how we encode values, resulting in inconsistencies in the way we encode metadata.

Problems only become worse when you start to think about searching across multiple files in this format. Another institution might have its own tabular data which contains relevant information for you, but how could you possibly perform a query across independent flat files in a consistent manner? Proficient users of Microsoft Excel could make use of macros and look-up tables to create links across multiple independent files, but these functionalities cannot be used outside Excel. This implies that you no longer have a platform- and application-independent format.

2.4 Change

How can tabular data evolve through time? The structure of catalogues and inventories does not change every month, but we could easily imagine at some point that we need to encode extra information, such as the technique (oil painting, aquarelle, etc.). Within the context of tabular data, we can simply add an extra column describing this new feature of the resources we are

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

documenting. If for some reason a column is no longer used or no longer contains relevant information, it can be deleted without any consequences for the rest of the data. Adding or deleting a column does not require you to make any modifications in the structure of the file. In this regard, an information system based on tabular data is resilient to change.

2.5 Implementation

Tabular data is one of the easiest conceptual formats, and as such, any software package will offer support. The most common form are *spreadsheet* applications such as Microsoft Excel, which in essence offer one giant table that can be modified. All spreadsheet software offers the possibility to export to TSV or CSV, albeit of course with the loss of formulas (cells that are calculated based on other cells), formatting (such as colour and borders) and functionalities such as macros, which we mentioned previously. Data types are also lost: all cell contents are stored as text.

Even with the simplest data model, a lot can go wrong in practice. Several elements are noteworthy here:

- Data can be separated by a *comma* but depending on a system's local settings, this might actually be a semicolon! For instance, in many European languages, a comma instead of a dot is used as decimal separator in numbers (so `1,5` is actually 1½). On these systems, it would thus be impractical to use a comma as column separator, hence the choice of a semicolon – so CSV is not always true to its name. In practice, CSV has come to stand for any separator-delimited type, which confusingly also includes TSV.
- Rows end with a *line break*. Unfortunately, different systems can produce different results. For instance, on Windows systems, a line break actually consists of two characters (a *carriage return* followed by a *newline*), whereas on Linux-based systems, it is just a single character (only a *newline*). Additionally, Linux-based systems might expect the last line to end with a line break, while this is not necessary on Windows.
- There is no way to indicate the difference between the *header row* and the rest of the data. This means that we will have to tell this explicitly to the parser.
- If the field value itself contains a comma or a line break, such as `Koons, Jeff` here, the value is typically enclosed in *double quotes*, so it can be parsed correctly as a single value and not as multiple rows or columns. Note that not all parsers support both cases; line breaks, especially, might be confusing. In general, enclosing any field with double quotes is allowed, even if no special characters occur within the value.
- Another dangerous issue is *character encoding*. Different systems use

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

different byte codes to represent characters, in particular if these characters lie outside the traditional ASCII alphabet, such as accented letters or Japanese characters. If one system has written a file in a certain encoding, it is important for another system to use the same encoding to read the file. Otherwise, an accented character in one encoding might accidentally be transformed into one or more other characters in a different encoding. This phenomenon is called *mojibake*, the incorrect presentation of characters due to an encoding mismatch. Chapter 3 will explore how the above-mentioned issues impact metadata quality and what can be done to mitigate them.

- What if the field value contains a double quote? This is solved by *escaping* the quote, adding a character in front of it that signals the next character has no special meaning. In the case of CSV, this escaping is done by doubling the quote. For instance, the value `width: 7", height: 5"` is encoded as `"width: 7"", height: 5"""`, wherein each literal quote is preceded by another.
- The main problem with CSV is that there are many ways to *encode* a file. The Internet Engineering Task Force (IETF) proposed a standard way of serializing CSV (Shafranovich, 2005), and this format can be read by most parsers. However, this by no means implies that all generators will follow this standard. Fortunately, most parsers are *adaptive*: they apply a heuristic on the file in question to determine which conventions were used. For instance, if every line in the file contains an equal number of semicolons, it is likely to assume that the delimiter is a semicolon. Also, if the third column always consists of decimal digits, except the first row (as in our example), then it can be assumed that the first line contains header data. Of course, none of these strategies are perfect; in practice, human verification is necessary for correct parsing.

3 Relational model

The relational model was developed to deal with the issues related to redundancies and inconsistencies as described above. Developed at the end of the 1970s, the relational model has been by far the most successful approach for managing structured data, and will continue to be used in the decades to come.

3.1 Model

The model asks you to take a step back from the individual metadata recorded in the tabular format and to identify on a higher level what the different *entities* are in the reality that you want to represent. We may define an entity as the ‘type of information that varies independently of another’ (Ramsay, 2004). Each

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

entity is characterized through the use of *attributes*. As depicted in Figure 2.1, entities are connected through *relations* to one another. Every record contains a unique *key* per table, which other records can use to refer to it in a relation.

3.2 Design methodology

Building a relational model is a difficult task that requires a lot of experience. Nonetheless, there are some guiding principles that you can use:

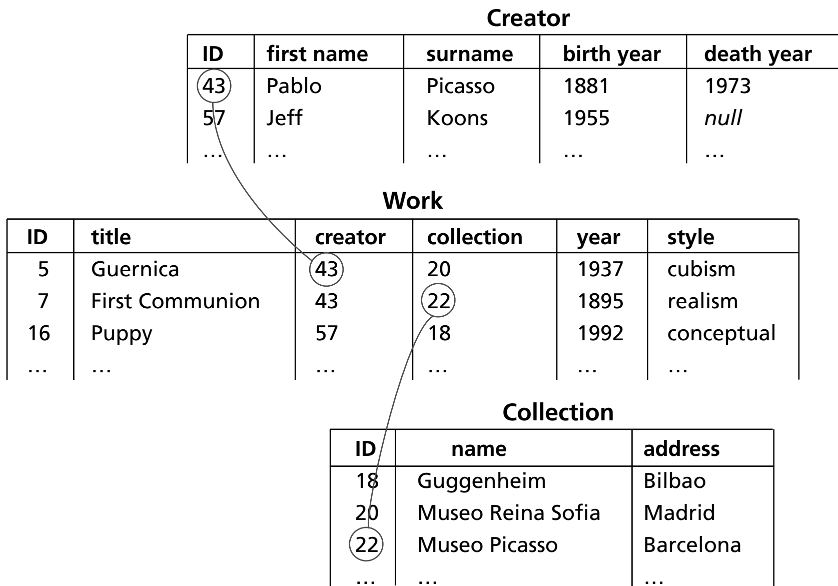
- The first task is to discover the *entity types* that the database will contain. Typically, entities correspond to independent concepts in the world of which there will be many, and each one has properties of its own.
  In our example, it is certain that 'Work' will be an entity type, as the database will contain several works. It is also likely that 'Creator' will be an entity type, as it is independent from Work and there will be many of them. However, an entity type of 'Country' will probably not be necessary, as we will only need the country's name and no other properties. However, for other use cases, it might be meaningful to encode 'Country' as an entity type.
- For every entity type, a table will be created in the database. Each row in the table will have a unique identifier, often a numeric value that is automatically generated. Each property of the entity will be a column in the table, and each column can have a value type. Our 'Work' table might have a textual field for the title and a date field for the creation date (or a four-digit field if we only plan to store the year).
- Next, *one-to-many relationships* must be modelled. They provide a mechanism for a record in a table to link to one record in another table, and the record in the second table can receive several such incoming links. As each entity has a unique identifier, we can add a column to the first table that will contain this identifier. That way, the objects in the first table can point to an item in the second table. For instance, the 'Work' table should contain a 'Creator' column that stores the identifier of the creator. That way, each work can be associated with one creator, and each creator can then be associated with several works.
- Finally, many-to-many relationships should be described. As fields in databases are traditionally not multi-valued, we must find another way for a record in a table to point to several records in another table. This is done by having a third relationship table, which has one column for identifiers of the first table and one column for identifiers of the second table. That way, we can find all items that belong together by traversing the rows of this table. Additional columns might describe properties of the relationship. In case works are authored by many creators, we could opt to represent them as a

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

many-to-many relationship. Then, we would have a third table called 'Creatorship' with columns 'work' and 'creator' that store the respective identifiers. We might add a textual column 'role' describing how the person was involved in the creation of the work.

The above points are merely guidelines. In practice, there are many possible motivations behind the choice for a certain design decision. There will always be trade-offs between optimal modelling flexibility, performance and simplicity. This is well illustrated by the option of whether to allow multiple creators for a work. This might allow you to describe the reality more closely, at the expense of a more complex (and possibly slower) data model.

Let us come back to our example. In order to have a sufficiently complex scheme, extra attributes, such as style, were added, as in Figure 2.2.
It should be clear that there is no such thing as one unique and perfectly adequate model, as the same reality can be interpreted in multiple ways.



Creator

ID	first name	surname	birth year	death year
43	Pablo	Picasso	1881	1973
57	Jeff	Koons	1955	*null*
…	…	…	…	…

Work

ID	title	creator	collection	year	style
5	Guernica	43	20	1937	cubism
7	First Communion	43	22	1895	realism
16	Puppy	57	18	1992	conceptual
…	…	…	…	…	…

Collection

ID	name	address
18	Guggenheim	Bilbao
20	Museo Reina Sofia	Madrid
22	Museo Picasso	Barcelona
…	…	…

Figure 2.2 An example entity relation model with the relationships highlighted

Depending on the importance you give to an aspect of the reality you are modelling, you either decide to consider it as an entity or as an attribute. Despite the simplicity of the scheme in Figure 2.2, many readers of this book probably would come up with different versions of the scheme. For example, one could decide to reduce the entity Collection to an attribute of the entity Work, if you do not consider the address as an independent aspect that needs to be

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

documented. The process of placing separate entities in separate tables is called *normalization*. Unfortunately, the more tables that need to be accessed to reconstruct the related metadata of an item, the slower operations will become. Therefore, a meaningful balance should be found between what data will be stored in a single table and what data is stored as a relationship.

The added complexity on the modelling level is made up for by the advantages offered by having a single record for an entity, that can be referred to with a unique ID. For example, every time you need to refer to the fact that an object is housed in a specific collection, you do not have to re-encode the metadata in relation to the address of where the collection is managed and other attributes of the entity Collection. You simply refer to the ID of the collection, and the same applies to other entities such as Creator. This approach ensures a lot more consistency. Those IDs are typically *indexed* to ensure the corresponding rows can be fetched in a fast way without traversing the entire table.

3.3 Implementation

Software built on top of this model, referred to as *relational database management software* (RDMS), has been extensively developed over the last decades and is currently at a very mature stage. Everyone has probably heard at some point about MySQL, the most popular open-source RDMS used for web applications, or MS-SQL, a proprietary RDMS developed by Microsoft. Another well known manufacturer is Oracle. Collection management systems, archival inventories and library catalogues are all built on top of a RDMS.

Most database systems work in a client-server set-up, where the server runs a RDMS and the client interacts with it using SQL statements. Consumer and small business applications, such as Microsoft Access, simplify the structure by offering a graphical user interface that works directly on top of a local database file. For regular RDMS, graphical interfaces for clients exist as well, but the communication underneath is done in SQL.

For instance, a table in MySQL can be created with:

```
CREATE TABLE Work (
  id INT AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(100),
  creator INT,
  collection INT,
  year CHAR(4),
  style VARCHAR(40)
);
```

This makes a new table called `Work` with an integer `id` column (`INT`), a textual

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

`title` column (`VARCHAR(100)`, meaning a string of characters with variable length, maximum 100), and integer `creator` and `collection` columns a 4-character `year` column, and a 40-character `style` column. The `id` column is special, since it should provide a unique identifier for each record. Therefore, it has the labels `AUTO_INCREMENT` (so new numbers are assigned automatically) and `PRIMARY KEY` (so the database knows that this is a unique field).

To insert data in this table, we can use:

```
INSERT INTO Work (title, creator, collection, year, style)
         VALUES ('Guernica', 43, 20, 1937, 'Cubism');
```

We supply the table name, followed by the names of the fields and then the values for these fields. Strings are surrounded by single quotes (and single quotes within strings are escaped by a backslash). Note how we did not supply a value for the `id` field, as this value is automatically generated (and will default to `1`, `2`, `3`, … on an empty table).

3.4 Search and retrieval

After several records have been inserted, we can retrieve them with `SELECT` queries. For instance:

```
SELECT * FROM Work;
```

will select all records in the `Work` table. If we only want the titles of works by Picasso (assuming he has identifier 43 in the `Creator` table), we can do:

```
SELECT title FROM Work WHERE creator=43;
```

This is only a basic introduction to SQL, as end-users are only confronted with predefined SQL queries accessible through a graphical interface of a collection management system. However, it is important to understand through the example the logic behind the SQL query language. Section 5 will build further on this example to illustrate how the SPARQL query language works.

3.5 Change

The previous section on the tabular format described how little impact change has on the structure of a flat file. Adding new columns or deleting existing ones does not fundamentally alter how the tabular data can be used. The situation could not be more different with relational databases. Adding an extra table requires the database manager to rethink the entire schema of the database, as

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

adding an extra table might imply a degrading of the normalization process.

Let us come back to our example. Imagine we want to add an entity *ArchivalItem* which describes the archival holdings of an artist that the institution possesses, such as correspondence, notes, personal photographs, historical press clippings, etc. How do we update our database with minimal effort? We can create a table *ArchivalItem* with the attributes *ID*, *document type*, *year*, *creator*, and *collection*. Then the change can happen by just adding this single table. However, that table would carry a considerable overlap with *Work*, as both have a creator, collection, and year. This information spread becomes difficult to manage in the end. If we want to respect the normalization requirements, we cannot just add extra tables, but we also need to modify the existing tables. Figure 2.3 shows a better integrated solution: the common attributes of *Work* and *ArchivalItem* are placed into a separate *Asset* table. However, this means that the existing table and data structure has to be modified.

Apart from ensuring the normalization of the new database schema, the

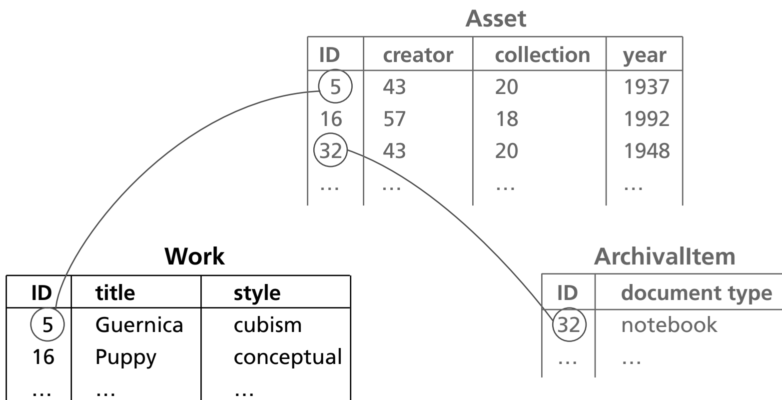


Figure 2.3 To support archival items in a consistent way, the *creator*, *collection*, and *year* fields must move from the *Work* table into a shared *Asset* table

modifications also impact external systems, such as the public front-end built to give access to the data on the web. Performing these types of updates and modifications every couple of months can be very cumbersome. In practice, these modifications are often avoided, as there is no time to fundamentally rethink the structure of the database. In this context, people often rely on lightweight and ad hoc solutions, such as creating a standalone spreadsheet. This type of short-term decision causes, over a period of years, tremendous issues with data consistency, as reference data are scattered across different applications. We can

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

therefore conclude that it is not a trivial matter to update and maintain a database, due to the complexity of modifying the database schema.

3.6 Sharing

As referred to in the beginning of this chapter, institutions already thought about interoperability between collections right from the start when RDMS were implemented in some pioneering cultural heritage institutions. There was (and there still is) a strong belief that acquiring the same collection management system provides the needed basis for interoperability. However, the customization of these software tools to accommodate specific requirements of each institution more often than not resulted in different approaches regarding the use of metadata elements. This made the exchange of records between institutions, which might have been using exactly the same software, problematic. Letting databases talk to one another and share their content is a complex matter, regardless of the application domain.

At this stage, it is important to point out the difference between *binary* and *non-binary* files. The previous section illustrated how tabular file formats make use of text files, allowing you to open a .tsv or .csv file with any standard text editor or spreadsheet software, making the exchange of metadata very straightforward. Databases, however, are stored in binary files which introduce a dependency on a specific software application. If you wanted for example to re-use a database of an institution, you would be obliged to use the same RDMS. Licences for proprietary RDMS easily cost around US$10,000 and you could potentially run into compatibility problems if you used different versions of the same software. You could also create a Structured Query Language (SQL) query, allowing you to create a data dump, and to import it afterwards in another application. But even if a standardized version of SQL exists, be aware that vendors implement the standard in varying ways. Certain RDMS have their own proprietary extensions, for example for column types, leading again to potential data compatibility problems.

We can therefore conclude that the interoperability of databases is quite problematic. Fortunately, methods have been developed to facilitate the export and import of structured data from and into different databases.

4 Meta-markup languages

Before we get into the details of how XML is used to facilitate the exchange of structured data, this section will make quite an extensive detour to the origins of markup and meta-markup languages. XML is probably the most abused and incorrectly used acronym (apart from RDF) at meetings in the cultural heritage sector. Some people consider it a programming language, others think it will automatically make their metadata *smart* and *semantic*. A broader view on the

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

origins of XML will allow you to understand the tremendously important difference between applying *markup* and *makeup*. Understanding the difference between a *data*- and a *narrative-centric* view of XML will also allow you to better understand why tool or standard A is better than tool or standard B, depending on whether you are managing text or data. Moreover, the evolution of XML is very much intertwined with the development and the future of HTML. The relevance of initiatives such as Schema.org or the OpenGraph protocol will also be better apprehended with a good understanding of the global context of meta-markup languages.

4.1 Adding structure to content

In parallel with the work on the development of relational databases for the management of structured data, producers of large volumes of unstructured texts, such as the pharmaceutical or aeronautics industry, developed the concept of a meta-markup language throughout the 1970s and 1980s. These industries are confronted with the need to manage complex and voluminous documentation of production and safety guidelines. In order to streamline the typesetting of these complex text documents, the idea was developed to make use of *markup* to indicate the presence of *structural elements* (title, subtitle, paragraph, etc.) inside a document.

Markup can be thought of as *annotations* added to a document. A manuscript would be annotated with signs indicating how specific parts of the text should be displayed. Through the use of *delimiters* (remember the role these play within the tabular model), such as angle (<>) or square brackets (`[]`), the markup is clearly distinguished from the text itself. The characters used to indicate the markup are purely a matter of convention, one could also use other characters such as `$` or `*`.

For example, if a specific string of characters which represents the title of a section should be printed in a large bold font, you could have the following HTML markup:

```
<font size="6"><b> Introduction to metadata</b></font>
```

Your browser would then render the text as:

Introduction to metadata

The above example is a typical illustration of how *markup* is merely used as *makeup*. The markup simply indicates how one particular string of characters, in our example ‘Introduction to metadata’, should be presented. Imagine you have a document containing several hundreds of section titles. Instead of

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

presenting them in size 20, you actually prefer to have them slightly bigger. Making this modification with the above approach obliges you to manually update the markup for every section title in your document.

Conceptually, a whole new world of opportunities for automated processing appears when the markup focuses on the *function* of a specific string of characters within the structure of a document. Instead of hardwiring how every individual element of a text should be presented, the markup can indicate the role it plays within a text. Let us re-use the same example and apply this time genuine markup and not makeup:

```
<h1>Introduction to metadata</h1>
```

We no longer indicate how the string of characters 'Introduction to metadata' should be printed. Instead, we specify the role this string plays within the text, by stipulating that 'Introduction to metadata' is the title of a section. You can re-use the markup element `h1` for all the titles of sections within the document. You only specify once how this specific structural element of your text should be formatted. This specification can take place either in the header of the document file or in a separate file linked to your document, which could contain the following definition:

```
h1 {
  font-size: 20pt;
  font-weight: bold;
}
```

This gives you the tremendous advantage that you can define in one central place how a specific structural element within your document should be displayed, from where it will then be implemented in a coherent manner across the entire document. Once the definition of the layout of a document is contained within a separate file, one can imagine having multiple files linked to a document in order to automatically switch between different designs. This is the idea behind the 'write once, publish many' principle. In a web context, it lets you automatically switch between a website with bigger or smaller fonts, or a standard version of a web page packed with images and colours and a Spartan page optimized for printing in black and white. The content stays the same, you just use another style sheet which indicates how the different elements of the web page should be rendered.

4.2 Model

Now that you understand the purpose of markup, we need to conceptualize its

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

use. In order to make markup machine-processable, you cannot just randomly put commands contained within delimiters inside a document. The markup needs to respect a logical and consistent structure to be processed automatically.

Conceptually, you can think of marked-up documents as trees. They have one *root* and consist of branches which themselves contain smaller branches, as depicted in Figure 2.1. A node and its directly descending nodes have a *parent–child* relationship; all directly descending nodes of a parent are *siblings*. The hierarchical nature of this data model is central: child elements inherit by default all of the characteristics which have been predefined on the level of the parent element. However, this default inheritance can be overridden if a specific characteristic is defined on the level of the child element.

4.3 Meta-markup

Why have we called this section 'meta-' markup languages? The model described above asks you to define a hierarchy of structural elements which you could consider as the building blocks of the documents you manage. Thinking about books, one could easily say that the element 'book' represents the root level of the document, which encloses all other elements. A child element of the root would be 'chapter', which itself consists of a title and multiple sections. We could imagine that this standardized vision of a book could be re-used by many people. But perhaps you like to use an epigraph at the beginning of every chapter, whereas other people would never make use of this element.

The early developers of meta-markup in the 1960s and 1970s foresaw that different application domains would have very different needs for the structural elements of their documents. For example, stanzas are an important structural building block of classical poetry. The documentation of production phases and testing of drugs might have specific elements which structure the quality procedures that need to be respected during the development of a drug.

It was therefore decided not to predefine all potentially interesting markup elements. Instead, a syntax and grammar was developed which allows everyone to develop their own specific markup language. Hence the use of *meta*, which refers to something at a higher, more abstract level. Out of the idea of a meta-markup language, the *Standard Generalized Markup Language* (SGML) was born. In hindsight, the heritage and impact of SGML, adopted as an ISO standard in 1986, has been enormous. SGML was used as a conceptual foundation for all the major standards which made the web a success: HTML, XML and CSS. Ironically, SGML has been considered by most as a failure, as the industry never largely adopted the standard due to its complexity. Bob Boiko's metaphor 'SGML became like those backwoods blues players of old to whom the pop stars give honor but no money' sums up the situation quite correctly (Boiko, 2005). The use of SGML required a thorough analysis of the

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

domain and content to be represented, followed by an intensive modelling exercise in which all essential structural elements of a document had to be predefined in a schema. The implementation itself was terribly expensive, due to costly software and the need for highly specialized staff. The success of HTML can be linked to exactly the opposite conditions: anyone can write HTML and the tools are freely available.

4.4 The Hypertext Markup Language

Around 1990, Tim Berners-Lee developed and implemented HTML. SGML was a major source of inspiration, but for reasons of simplicity a fixed set of elements was defined, representing the basic building blocks of a web page (e.g. `<head>`, `<title>`, `<body>`, `<link>`). Notice how these elements indicate structural elements of a web document, and do not stipulate any layout. This markup is to be parsed by a web browser, which is responsible for interpreting the HTML tags and for displaying the web document on the computer screen. HTML is therefore a markup language (and not a meta-markup language). This implies that you can only make use of a pre-defined set of tags which can be interpreted by a web browser. Nothing holds you back from inventing your own HTML tags, but in order to use them you would need to build your own browser.

Needless to say, HTML was a success. However, after a decade Berners-Lee's brainchild was corrupted from a markup into a makeup language. The focus on the aesthetics (and not on the semantics or structure) of web pages was beginning to undermine the potential of the web as a global information system. What happened?

From the mid to the end of the 1990s, web publishers were building up the dot com bubble. During this period, one of the biggest business model underlying the web was born. Within this model, the value of a company does not lie in its net income acquired through the commercialization of a product or a service offered to its user base. The mission of a company is to rapidly build up a user base by offering a free commodity (email, social networking, photo sharing, etc.). This business model is based on the assumption that the company will be able to monetize its customer base at a latter stage through advertising and the aggregation of consumer profiles for example.

With this in mind, it is easy to understand why web developers focused in the first place on an attractive and distinctive layout. This tendency played an important role in the browser war between Netscape Navigator (precursor of Mozilla Firefox) and Microsoft Internet Explorer. In order to attract the biggest user base, both browsers developed, independently one from the other, HTML elements that would render web content attractive and original. To fully understand the impact of these practices, do the following small exercise. Launch Notepad or any other simple text editor and encode the following HTML:

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

```
<html>
<blink>This text only blinks in Firefox</blink>
</html>
```

Make sure to save the document with the .html extension. Now start up the Firefox web browser and open the HTML document you just created. Congratulations, you have just created your first makeup'ed web page: the text contained within the `<blink></blink>` tags should blink. Now open up the same document in any other web browser (Microsoft Internet Explorer, Google Chrome, Apple Safari...). Nothing happens. The browser understood that there are tags but does not understand them and therefore just displays the text contained within them.

The blink element is a non-standard presentational HTML element introduced in Netscape Navigator, but not supported by other browsers. Anyone who is old enough to have surfed the web in the late 1990s will think fondly of all the weird and utterly user-unfriendly websites which held flashing and hovering content. On top of that: exactly the same page displayed differently across browsers.

4.5 The eXtensible Markup Language (XML)

The interoperability issues described above, coupled with the exploding volume of HTML documents containing no exploitable structure or semantics, resulted in a growing unease within the information retrieval and information science community. A standard was needed to ensure a more structured web, and XML saw the light. The standard is built as an application profile of SGML, but simplifies its use. An effort was made to keep 80% of SGML's functionality with only 20% of its complexity.

XML being a meta-markup language, realizes that user communities have the possibility of defining their own markup elements, hence the adjective 'extensible' in the name of XML. The big advantage of XML, especially at the time of major incompatibility issues on the web, is its platform and application independence. With its open and standardized format, XML allowed the web community to make a big step forward with the publication of structured content.

4.6 Designing XML documents

Similarly to relational database schemas, the design of XML documents also involves extensibility and simplicity trade-offs. The main discussion in XML is whether to model an entity as an element (serialized as *tags* surrounded by angle brackets) or as an *attribute* (key/value modifiers of a tag). Every XML document begins with an XML declaration, a *processing instruction* that identifies the document as a specific XML version. Processing instructions are special tags that

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

start and end with questions marks inside the angle brackets. For instance, a minimal XML document for our collection would be:

```
<?xml version="1.0" encoding="UTF-8"?>
<Art title="Modern art"/>
```

So what we see here is the XML declaration, followed by an `Art` tag that has a `title` attribute with value `Modern art`. The `Art` element is the *root element* of our document, and every XML document should have exactly one root element. Since there are no other elements yet, we have made `Art` *self-closing* by including a slash before its ending angle bracket. Note how we were free to choose the names of the tag and the attribute, in contrast to more specified languages such as HTML. That does however not imply total freedom: the mandatory `version` attribute and its value are predetermined by the XML standard. The `encoding` attribute is not mandatory, but as we said before when explaining tabular data, plain text files always have a risk of being interpreted in a different encoding from that intended. Therefore, by specifying the encoding, we ensure that the interpretation will happen uniformly.

The root element is not difficult to get right, but modelling questions arise when we add data elements. For instance, let's add a work to the collection.

```
<?xml version="1.0" encoding="UTF-8"?>
<Art title="Modern art">
  <Work title="Guernica" year="1937" creator="Pablo Picasso"
        collection="Museo Reina Sofia" location="Madrid"/>
</Art>
```

The *hierarchical structure* of XML documents now becomes apparent: the `Work` element is a child of the `Art` element. Initially, we have chosen to model the work's elements as properties. However, this might not prove extensible enough. For instance, it is difficult to add more structure to the `creator` field, and there is currently no relation between the `collection` and `location` fields. The opposite approach would be to model everything as child elements:

```
<Art title="Modern art">
  <Work>
    <Title>Guernica</Title>
    <CreationDate>
      <Year>1937</Year>
    </CreationDate>
    <Creator>
      <FirstName>Pablo</FirstName>
```

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

```
      <LastName>Picasso</LastName>
    </Creator>
    <Collection>
      <Name>Museo Reina Sofia</Name>
      <Location>Madrid</Location>
    </Collection>
  </Work>
</Art>
```

This leaves us maximum flexibility to extend the document at any point. However, this also comes at a cost: the hierarchy is now relatively deep to express simple concepts, even for straightforward properties such as a year of creation. Even though the original design goals for XML state that 'terseness in XML markup is of minimal importance', it might be important for our application. Although software does not have any more difficulty parsing hierarchies as opposed to attributes, unnecessary complexity is never an asset. Understanding the XML document at a glance becomes more difficult for humans (and XML was designed to be read by both humans and machines), and the job of programming the format reader on top of the XML parser becomes more complex. In practice, a compromise often works best:

```
<Art title="Modern art">
  <Work title="Guernica" year="1937">
    <Creator firstName="Pablo" lastName="Picasso"/>
    <Collection name="Museo Reina Sofia" location="Madrid"/>
  </Work>
</Art>
```

Here, we have chosen to model all values that will not be decomposed or require further properties as attributes. For instance, a work's `title` does not require further description, but we would add additional information to a `Creator`, such as date and place of birth.

This might remind you of the discussion on relational model design, where we first determined *entity types*. Indeed, the decision is similar: things that would end up as entities in databases are likely represented as elements in an XML document as well. In contrast with databases, XML documents are more flexible and there is an even larger grey area for modelling choices. As always, this extended flexibility comes at a cost: databases are made for rapid data search and manipulation; searching XML documents is more than an order of magnitude slower.

Speaking of databases, you might wonder how to represent relations in XML. The answer is that you are free to choose that, but some choices are wiser than others.

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

For instance, we can simply continue the model as above and add another work:

```
<Art title="Modern art">
  <Work title="Guernica" year="1937">
    <Creator firstName="Pablo" lastName="Picasso"/>
    <Collection name="Museo Reina Sofia" location="Madrid"/>
  </Work>
  <Work title="First Communion" year="1895">
    <Creator firstName="Pablo" lastName="Picaso"/>
    <Collection name="Museo Picasso" location="Barcelona"/>
  </Work>
<Art>
```

However, this duplication of information is harder to maintain and it might lead to errors. In fact, the last name of the creator of the second work is incorrectly spelled 'Picaso', even though it appears correctly in the first. Therefore, it makes sense to model the information only once and refer to it using identifiers:

```
<Art title="Modern art">
  <Work title="Guernica" year="1937" collectionId="Co20">
    <CreatorRef creatorId="Cr43"/>
  </Work>
  <Work title="First Communion" year="1895" collectionId="Co22">
    <CreatorRef creatorId="Cr43"/>
  </Work>
  <Creator id="Cr43" firstName="Pablo" lastName="Picasso"/>
  <Collection id="Co20" name="Museo Reina Sofia" location="Madrid"/>
  <Collection id="Co22" name="Museo Picasso" location="Barcelona"/>
</Art>
```

In contrast to database systems, you are responsible yourself for the correct assignment and use of identifiers. Note how we modelled `collectionId` as an attribute of `work`, but `Creator` as a child element. The rationale behind that is that a work only resides in one collection, whereas there might be many creators of a single work, and a separate element allows us to specify a role for each of them (as with the many-to-many relationship of a database schema). This design choice allows the specification of multiple creators, since attribute names on an element must be unique.

4.7 XML Schema

The flexibility of XML documents might seem a drawback if you want to

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

consume XML. After all, your application will expect to see specific elements and attributes, but if anybody has the freedom to create their own, how can you be sure that the things you need will be there? We briefly mentioned before that this is possible with an XML schema, a document that explains what kind of XML markup is allowed.

Different languages exist to express schemas, the oldest being Document Type Definition (DTD), part of the original XML specification. The DTD specification of our `Work` element might look like this:

```
<!ELEMENT Work(CreatorRef)+>
<!ATTLIST Work title CDATA #REQUIRED>
<!ATTLIST Work year CDATA #REQUIRED>
<!ATTLIST Work collectionId IDREF #REQUIRED>
```

We again note a special kind of tag, which starts with an exclamation mark. The above fragment states that `Work` is an element that can contain many `CreatorRef` elements. It can have `title` and `year` attributes of type `CDATA` (character data) and a `collectionId` attribute that is an `IDREF` (a reference to an identifier), all of which are `REQUIRED`. This allows a parser to check whether the `Work` element is specified in the right way. Additionally, it can verify whether the identifiers are used correctly, as it will check for each `IDREF` attribute whether an element with this ID exists.

However, DTD has a quite peculiar syntax and it does not have a strong expressive power. For instance, we could not specify that `year` is a numeric value. Also, more complicated hierarchical rules cannot be efficiently described. Therefore, XML Schema (note the capital 'S') has been created by W3C (the World Wide Web Consortium). It features an XML syntax to describe schema documents, which themselves can also be validated by XML Schema. A description of the `Work` element would be:

```
<xsd:element name="Work">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="CreatorRef" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="title" type="xsd:string"/>
    <xsd:attribute name="year" type="xsd:gYear"/>
    <xsd:attribute name="collectionId" type="xsd:IDREF"/>
  </xsd:complexType>
</xsd:element>
```

This says that `Work` is an element that can have several `CreatorRef` elements. It

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

can have a `title` string as attribute, a `year` that has a *year* data type, and `collectionId` which is an `IDREF`. We see that the XML Schema syntax is more verbose, but it is also more expressive. For instance, the `year` field is now specified more precisely thanks to XML Schema built-in data types.

For documents with an associated DTD or XML Schema, various automated *validators* exist that either guarantee the validity of a document or show what type of errors occur. Many software libraries for XML parsing support this functionality. Checking the validity of an XML document upfront means the rest of your software chain can read and manipulate the document as expected, without causing errors because of missing or incorrect structure.

4.8 Namespaces

As anyone can make their own elements and attributes in an XML document, we need a mechanism to universally identify which ones are the same. For instance, two documents might use a `title` element, but one uses it to designate book titles, and the other for personal titles such as `Mr` or `Mrs`. While enforcing a specific document structure, schema documents alone do not provide a means for consistent re-use across *different* types of documents that need to re-use the *same* elements in another context.

This is the issue that *XML namespaces* address – they are a method of qualifying element and attribute names (Bray et al., 2006). Namespaces allow you to re-use what has already been developed by someone else, and by doing so you can explicitly state that you agree with outside parties on how your data should be interpreted. The link with metadata schemes is self-evident here, as they share the same goal: making explicit statements about how a specific value should be interpreted. For example, if I want to use an element in my XML document which represents the name of a creator, it would make a lot of sense not to issue an identifier on my own for that element, but to re-use the namespace issued for the Dublin Core element "Creator": `http://purl.org/dc/terms`. When used on a `creator` element, it indicates that this element is to be interpreted as defined by the Dublin Core schema, which defines it as "an entity primarily responsible for making the resource".

Namespaces can be indicated using the reserved XML attribute `xmlns` on an element, which then holds for this element and all of its descendants. Namespace declarations are mostly seen in the `schema` element, forming the root of the schema, and are applied to the entire document. For instance, an XML document could start with the following tag:

```
<Agent xmlns="http://purl.org/dc/terms/">
```

This indicates that all elements in the document, including the root element, are

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

to be interpreted according to the Dublin Core specification, which defines this namespace. Multiple namespaces can be used in a single document by using prefixes. For instance, we could re-use elements from a generic schema such as Dublin Core in combination with more specific elements from VRA Core, as follows:

```
<Art title="Modern art"
     xmlns:dc="http://purl.org/dc/terms/"
     xmlns:vra="http://vraweb.org/vracore4.htm">
  <Work>
    <dc:creator>Pablo Picasso</dc:creator>
    <dc:title>Guernica</dc:title>
    <vra:technique>Oil painting</vra:technique>
  </Work>
</Art>
```

4.9 Search and retrieval

While relational databases are especially designed for maximal performance, XML documents are designed for maximal flexibility. As we have seen, information can be modelled in different ways. As such, we cannot expect XML to achieve the same level of speed for search and retrieval, even if the entire model is loaded into memory (which is not always possible, due to size constraints). Nonetheless, like databases that are accessible through SQL, XML has its own query language: XPath. Since XML is a tree, XPath allows us to traverse that tree and collect elements and attribute values along the way. The result of an XPath query is thus not an XML document, but a set of elements or values.

Given the structure of XML, it does not come as a surprise that XPath has a hierarchical division as well. For instance, the following XPath query selects all `Creator` elements that are children of any `Work` elements in an `Art` document:

```
/Art/Work/Creator
```

So we first select `Art`, the root element (note the leading slash `/`), then all possible `Work` children, and finally all `Creator` elements that are direct descendants thereof. To select `LastName` elements that are children of any `Creator` element, we can use:

```
Creator/LastName
```

Note how the XPath expression does not start with a slash this time, as we do not start from the root but rather from any possible `Creator` element. However,

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

this will only select children, i.e., direct descendants, of `Creator`. To select *all* descendants of a node, we can specify an *axis* called `descendant`:

```
Work/descendant::LastName
```

This will find all `LastName` elements that are somewhere inside a `Work`, even if nested within other elements.

Finally, this expression selects all `year` attributes from `Work` elements:

```
Work/@year
```

Many more constructs are possible. We can filter elements based on attribute values or the elements they contain, much as you would expect from SQL queries. Bear in mind that XPath queries are executed by traversing the whole XML tree, in contrast with relational databases, which use indexes of the data.

4.10 Data- versus narrative-centric XML

One of the reasons why there are so many misconceptions about XML is the fact that it can be used for a wide range of purposes. Even if there are a lot of concrete projects and applications which do not fall exclusively in one of the two categories, it is important to distinguish the data- and narrative-centric approach to XML. These two categories largely coincide with how the two different communities we talked about in the beginning of the section understand XML. The IT community has a data-centric view, in the sense that XML is used to define a structure, which is then filled up with data. The example of a Simple Object Access Protocol (SOAP) message below illustrates this approach. The XML file allows to facilitate, in an automated manner, the communication of a specific value between two computers, here the insurance value of Guernica at the Reina Sofia Museum, in a structured format.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header/>
  <soap:Body>
    <m:GetInsuranceValue xmlns:
       m="http://www.example.org/insurance_value">
      <m:Insurance_value>DE00050</m:Insurance_value>
    </m:GetInsuranceValue>
  </soap:Body>
</soap:Envelope>
```

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

The digital humanities and computational linguists tend to have a narrative-centric vision of XML, in the sense that XML is used to insert some level of structure in documents. The Text Encoding Initiative (TEI) is a classic example of the narrative-centric approach. Below you can find an excerpt from a TEI example file from Wikipedia. The choice tag can be used to represent variants of the same section of text. In the example below, `choice` is used to indicate an original and a corrected value and to differentiate an original and regularized spelling:

```
<p xml:id="p23">
  Lastly, That, upon his solemn oath
  to observe all the above articles,
  the said man-mountain shall have a daily allowance of
  meat and drink sufficient for the support of
  <choice>
    <sic>1724</sic>
    <corr>1728</corr>
  </choice> of our subjects,
  with free access to our royal person, and other marks of our
  <choice>
    <orig>favour</orig>
    <reg>favor</reg>
  </choice>.
</p>
```

The fact that XML can accommodate both approaches is due to the fact that XML was conceived to be both human- and machine-readable. This feature of XML is one of both its biggest advantages and its biggest pain points. Any XML file can be opened and modified in a text editor. In theory, you could describe your entire collection in all of its detail by just using Notepad. This platform and application independence makes it particularly easy to exchange XML documents between heterogeneous environments. XML files are in this sense one of the best serialization formats that make the case for non-binary files.

4.11 Change

Most changes in XML are difficult and should be carefully considered, as they need to propagate to different documents. In the case of relational databases, changing the structure was cumbersome, but nonetheless always limited to a single system. With XML, if a document format evolves, there are two main options:

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

- The change is completely *backwards-compatible*; for example, adding an optional element. Existing documents can then remain as-is, and parsers can be extended. However, this needs careful planning in advance, and is not always possible.
- The change is not backwards-compatible; for example, renaming a tag or changing an attribute into a child element. We distinguish two sub-options:
  - Through schema *versioning*, different document structures can be supported. Existing documents do not have to change, but parsers must support the different versions (for example, one with the old element name and one with the new name).
  - The change is *breaking* and existing documents and parsers have to be updated to conform to the new schema. This leads to maximal consistency, but many modifications must be carried out (for example, all documents and parsers have to switch to the new name).

The first option is clearly optimal, but only applies to certain cases. In general, change is difficult and thus progresses slowly. Between changes, data cannot be stored optimally.

4.12 Why do IT people prefer JSON?

According to the followers of the XML hype around the year 2000, XML was soon going to take over the web (and then the world). There are several reasons why this did not happen – even though XML is still very popular in many application domains. First, there is XML's verbosity. By design, XML tries to be as explicit as possible, but this sacrifices clarity in the end. For many XML documents, it is difficult to understand what it going on with a single look. There is also a lot of repetition in the markup.

More importantly, the web has witnessed an enormous growth of JavaScript applications. At first regarded as inferior to compiled languages such as Java (which has many XML-driven parts itself), it soon became clear that JavaScript's dynamic nature made it a perfect fit for the web. While JavaScript can parse XML, it also has a native format to express data: the JavaScript Object Notation or JSON, with a more terse syntax that focuses on 'just getting things done'. As a result, we should not expect much portability between different contexts. However, data exchange over the web between clients and servers, and between different applications, happens mostly in JSON. Severance (2012) neatly describes this oddity of the winning underdog. An example JSON fragment of a work could look like this:

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

```
{
  "title": "Guernica",
  "year": 1937,
  "creator": {
    "firstName": "Pablo",
    "lastName:" "Picasso"
  },
  "collection": {
    "name": "Museo Reina Sofia",
    "location": "Madrid"
  }
}
```

Note that JSON has also a hierarchical structure; in fact, the above document translates directly into XML (with the exception that JSON does not offer distinct structures for child element and simple attributes).

4.13 Does XML make your data smart?

In the introduction of this section we mentioned the high expectations collection holders have of XML, due to the common belief that XML makes your data *smart*. We hope to have demonstrated throughout the section that XML is nothing more, but also nothing less, than a standardized syntax to encode data in a structured manner. The semantics of data can be made explicit through the use of XML elements, but it is important to realize that outside the community which defined the elements, the meaning of the elements is not explicit. The use of namespaces does offer the opportunity to share amongst different communities the same element, but this practice mostly applies to a limited set of the elements. So at the end of the day, even with our metadata in a more portable format we are still confronted with the same problem: if we want other people to re-use our metadata, they are forced to study the schema and documentation describing their semantics.

5 Linked data

We have travelled all the way from previous data models to come to this specific point. Relational databases and XML both offer wonderful possibilities to manage structured metadata, but they also have the big drawback that you need to understand the schema describing the structure and interaction between the data. This is exactly where our last data model comes in.

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

5.1 The semantic web vision

Before we get into details regarding the data model, let us first fully understand why exactly we need to bypass the problems associated to the re-use of locally defined semantics. The vision behind the semantic web was born out of the frustration of having only human-readable information on the web, which restricts the ways in which software can help us find information. For instance, keyword-based search works well for terms such as 'Picasso'. Queries such as 'paintings by Picasso' are already more difficult, since pages can use different wording. But without an interpretation of a page's content, queries such as 'paintings by artists who have met Picasso' are impossible. In the semantic web vision, the web also becomes accessible for software agents instead of containing only human-readable information (Berners-Lee, Hendler and Lassila, 2001). It enables a vast array of novel applications by making information machine-interpretable.

5.2 RDF

By adopting an extremely simple data model consisting of *triples*, data represented in Resource Description Framework (RDF) become *schema-neutral*. An RDF triple consists of a *subject*, *predicate* and an *object*, as seen in Figure 2.1. This allows for maximum flexibility. Any resource in the world (the subject) can have a specific relationship (the predicate) to any resource in the world (the object). There is no limit on what can be connected to what. This model allows us to express statements in a straightforward way, such as for example the statement that Jeff Koons is the artist who created the work 'Puppy':

```
:Jeff_Koons :created :Puppy.
```

Figure 2.4 represents some of the metadata of the example we have been using throughout this chapter. By simplifying to a maximum the data model, all of the semantics are made explicit by the triple itself. By doing so, there is no longer a need for a schema to interpret the data. Within the world of databases and XML, only the data conforming to the rules defined in the schema may exist and be encoded in the database or XML file. With RDF, you just make statements about facts you know, but these statements might interact with statements made outside your information system. This data model allows heterogeneous data to connect and interact. For instance, in Figure 2.4 you can see two pieces of metadata which were previously not mentioned.

Note, however, that schema-neutral does not mean that no schema-related issues remain. Any piece of data still needs to be expressed in a certain vocabulary, and each vocabulary has its own way of expressing things. The main difference between RDF and XML and other technologies is that in RDF everything is self-describing: each vocabulary is expressed in terms of other

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

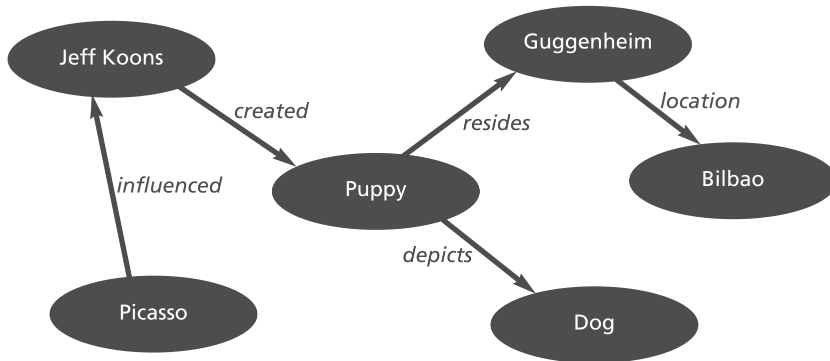


Figure 2.4 Illustration of how to use triples to express metadata

vocabularies. In order to extract meaning from a given RDF fragment, the unique identifiers of each used resource allow its definition to be looked up. This look-up mechanism is enabled by the principles of linked data, which is the topic of the next section.

5.3 The linked data principles

The implementation of the RDF model in the open and distributed context of the web is based upon their capability to issue identifiers for subjects, predicates and objects, which can be freely re-used. Software is then able to interpret this information, because the identifiers create unique meaning, as opposed to the names of columns in databases or elements in XML, which only have local significance and change from application to application.

However, the semantic web was mainly developed from the artificial intelligence (AI) standpoint. Ever since the 1960s, the AI community worked on automated reasoning, expert systems and intelligent agents. Underlying all of these fields and applications is a core belief in the power of logic to formalize all aspects contained within an information system. Chapter 4, in section 4, will come back to the reasons why this vision is currently deemed unworkable on the scale of the web.

In order to move forward with a machine-readable web, Berners-Lee (2006) drastically reduced the ambitions of the full-blown semantic web and came up with the *linked data principles*. These four rules specify a simple way to format data so it can be interpreted by software:

1 Use URIs as names for things.
2 Use HTTP URIs so that people can look up those names.
3 When someone looks up a URI, provide useful information, using the

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

standards (RDF, SPARQL).
4 Include links to other URIs so that they can discover more things.

As you can see, these principles require a clear understanding of URLs and URIs. A URL is a *uniform resource locator*, which, as the name says, enables to locate resources in a unique way. The most widely known URLs are those used on the web; they start with http: or https:. Given any such URL, your browser is able to locate the underlying resource, no matter where it is physically stored. A URI, *uniform resource identifier*, is a generalization of the concept that permits resources anywhere in the universe to be given a unique identification. However, not all URIs are URLs; for instance, the URI `urn:lex:eu:council:directive:2004-12-07;31` uniquely identifies a European Union directive, but does not directly give its location.

Let's now look at the role of URIs and URLs in the linked data principles. The first principle demands unique identification for each concept, and URIs are the most appropriate mechanism to provide this. Additionally, the second principle states that these identifiers must be HTTP URIs, in other words URLs on the web. The third principle asks for the representation of the resources identified by those URLs by using standards, such as the machine-readable format RDF. Finally, the fourth principle makes sure that data contains links to other data, allowing software agents to look up related information.

5.4 The central role of URLs

Remember how XML namespaces uniquely identified elements and attributes. With linked data, URLs are used to uniquely identify concepts. For example, a more meaningful way to express the fact that Jeff Koons created 'Puppy' is the following triple:

```
<http://guggenheim.org/new-york/collections/collection-
online/artwork/48>
    <http://purl.org/dc/terms/creator>
      <http://viaf.org/viaf/5035739>.
```

In this example you see that the artist Jeff Koons is identified with the URL of his authority file available on the Virtual International Authority File (VIAF) website. What is the added value of using the URL instead of the string of characters 'Koons, Jeff, 1955–'? Rules have been developed for decades to formalize the spelling of names in authority records, and in order to disambiguate with other people with exactly the same name, his date of birth has been added. Therefore, one could think that the text string serves well as an identifier. Imagine, however, what needs to happen if Jeff Koons dies in 2025? All of the

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

metadata which used the text string to designate the name of the creator will need to update the creator field to 'Koons, Jeff, 1955–2025'. Instead, if the VIAF URL is used, the information only needs to be updated centrally in the VIAF authority file, but the URL as such does not change. From the moment the date of death has been added, this new information will become automatically available to everyone who uses the VIAF URL as an identifier for Jeff Koons. The fact of looking up more information about a subject through its URL is called *dereferencing*. Chapter 5 will explain the use of URLs for both virtual and real-world resources.

The basic condition for this approach is a stability of the identifier, and URLs tend to have a very bad reputation on that level. The URL used to identify the work 'Puppy' is simply the URL of the record displaying the metadata of the object. But what would happen if this work is transferred from the Bilbao to the Venice Guggenheim museum? This would imply that the URL loses its validity. To avoid such trouble, Chapter 6 discusses sustainable URLs.

5.5 Serialization

As the initial semantic web vision was launched in 2001, it comes as no surprise that the first standardized syntax was based on XML, and consequently named RDF/XML. Unfortunately, RDF/XML inherits the verbosity of XML as well, resulting in a serialization format that admittedly can be parsed by an XML parser, but is hard to follow and understand. Therefore, the Turtle syntax was developed, in which triples are native elements. Turtle is currently in the final stage of standardization, and is bound to take the place of RDF/XML.

The triples above were expressed in Turtle, but here we will review its syntax in more detail. In Turtle, triples are serialized by separating each of the components (subject, predicate, object) by white space and ending it with a dot. URLs are surrounded by angle brackets.

Since URLs can be rather long, it includes an abbreviation mechanism through the `@prefix` directive:

```
@prefix gh: <http://guggenheim.org/new-york/collections/collection-
  online/artwork/>.
@prefix dc: <http://purl.org/dc/terms/>.
@prefix viaf: <http://viaf.org/viaf/>.

gh:48 dc:creator viaf:5035739.
```

We first define three *prefixes*, consisting of certain characters ended by a colon, which can subsequently be re-used. This makes the actual triples shorter and

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

easier to understand, and can also eliminate possible mistakes in the URL by avoiding duplication.

Multiple statements about the same object can be written tersely by using a semicolon if the subject is repeated, and a comma if the subject and predicate are repeated:

```
gh:48 dc:creator viaf:5035739;
      dc:title "Puppy".
viaf:5035739 :influencedBy viaf:15873,
                           viaf:95794725.
```

The above fragment states that the creator of the artwork is Jeff Koons (`viaf:5035739`) that its title is 'Puppy', and that Jeff Koons is influenced by Pablo Picasso (`viaf:15873`) and Ed Paschke (`viaf:95794725`). In addition to the use of semicolons and commas, we note two other things. First, the word 'Puppy' is surrounded by double quotes, as it is not a URL but a *literal value*. RDF includes literal values in its model as well, as some properties eventually do not point at another object but rather at a non-decomposable value. Literal values can have an associated *type* (such as string, number, or date), and in case of a string, a *language code* (such as `en-us`). Second, the predicate `:influencedBy` has an empty namespace prefix, which indicates that it is local to the current document. This is a convenient way of introducing new concepts in a document, which are then defined in terms of other properties later on.

5.6 Search and retrieval

Like relational databases and XML before, RDF also comes with its own query language, *SPARQL*. Queries in SPARQL are based on *graph patterns*: the form of the desired data is described in a `WHERE` clause. A simple SPARQL query is the following one:

```
SELECT ?predicate ?object WHERE {
  <http://dbpedia.org/resource/Pablo_Picasso> ?predicate ?object.
}
```

This searches for triples that have `Picasso` as subject and *any* predicate and *any* object. The question mark before a word means that it is a *variable*. Out of those triples, the query will return the predicate and the object.

We can be more specific as well. For instance, if we want to find works by Picasso, we can use the following query. Note the use of prefixes to abbreviate common terms.

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpprop: <http://dbpedia.org/property/>

SELECT ?work WHERE {
  ?work dbpprop:artist dbpedia:Pablo_Picasso.
}
```

This illustrates the simplicity of the linked data model, while at the same time showing its immense power and flexibility.

5.7 Change

We hope you understand an essential feature of the RDF data model by now: that all of the semantics of the data are made explicit through the model itself. In a sense, we have come full circle with a return to the idea of a flat file, if we think of a collection of triples contained in a single file, composed of three columns with the headers *subject*, *predicate* and *object*. Confronted with a new reality which needs to be handled, new triples are simply added. However, this comparison does not do justice to the RDF model, as the strength of triples is that every value points to other triples that have this value as subject or object. The context of every row is thereby augmented by other rows.

Change in RDF is therefore supported easily: adding new data comes down to adding new triples, without needing to alter the existing data or structure. This gives the data maximum flexibility, at the cost of dealing with an open world. Whereas databases are guaranteed to give you all the data they have, finding all facts about a concept is more complicated with RDF, as different identifiers might be used for the same thing. Nonetheless, when such issues are managed properly, for instance by creating sufficient links between datasets, schema-neutrality can be a very powerful concept.

6 Conclusion

This chapter provided an answer to the question of *why* we need linked data. The answer might seem self-evident and straightforward: in order to *link* data across the web. To achieve this goal, we need to be able to interconnect data across independent islands. We use the word ‘islands’, as each information system is modelled for its particular needs and application domain, resulting in systems that cannot hold hands with one another in an automated manner. For sure, it is easy to embed a link in your collection database which points to the record of a similar object from another institution. But this requires you to know how to access the database of the other institution, to know what fields are used to describe the object. Once you have found the record to which you want to link,

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

you need to embed its URL manually within the record of your database. We cannot reasonably perform these actions manually for all of our collection items. We therefore need to think about how we can automate the linking process.

6.1 Understanding trade-offs

In order to understand the obstacles to the automation of linking, this chapter gave a comprehensive overview of the most recurrent data models used to build the current islands of metadata. Hopefully the red thread between the four data models has appeared clearly. A *trade-off* has to be made between the complexity of the data model and the ease with which the outside world can re-use and connect to your data. The collection management databases which are currently forming the backbone of our cultural heritage institutions allow complex data to be stored in a way which minimizes redundancy and dependency. You only need to encode once all the information you have in relation to an artist or some very rare and complex technique which requires a lot of documentation to be understood. The day the artist dies, you only need to update the attribute 'Date of death' of the entity 'Artist', and this update will be shared across all the records pointing to this entity.

However, we have seen that this advantage comes at a cost. A collection management database can easily contain several hundreds of tables, interconnected with relations. Modifications and extra tables are often added in an ad hoc manner in order to fulfil an urgent need, and are often left undocumented. It should therefore come as no surprise that migration operations from one software to the other (or even just to another version of the same software) can be very time-consuming for IT staff, as they need to interpret the database schema to understand how the tables are interconnected.

The development of XML made it easier to share metadata across applications, due to its platform and application independence. In contrast to databases, you do not need any specific software to read and create XML documents. The advantage of being readable both for humans and machines also resulted in XML's major drawback, namely its verbosity. More importantly, complex data described in XML rely on a schema documenting and prescribing how elements and attributes interact within an XML file. Establishing a consensus inside and outside an institution on how to interpret and update the schema often causes problems.

The last section of this chapter introduced RDF, which we referred to as schema-neutral. The simplicity of the data model (subject-predicate-object) brings back the idea of a flat file, consisting of three columns representing subjects, objects and predicates. No extra documentation or schema is needed to interpret these data, and any new type of information can be added without a need to modify the structure of the data model.

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

6.2 The law of instruments

'Give a small boy a hammer, and he will find that everything he encounters needs pounding'.[2] This expression, also referred to as the 'law of the instrument', describes how people have a tendency to attribute too much importance to the tool they are using, at the expense of their objectives. One would think that engineers evaluate what data model most suits the needs of an application, and then choose a technology allowing them to implement the model. In practice, the opposite often happens. People build up experience with a specific technology and are not very eager to switch to another tool, as this sometimes requires a substantial effort. Academics and consultants, on the other hand, occasionally tend to get overly enthusiastic about a new technology, regardless of whether the underlying data model is best for the purpose at hand.

6.3 When to use what

This chapter hopefully made it clear that every model has been developed for a specific use. If the only tool you have is a hammer, you treat everything as if it were a nail. This was the case with the use of XML in the context of SOAP, for example. The verbosity of XML is now considered inadequate for data exchange over the web between clients and servers, and its role is taken over by JSON. The current hype on linked data reminds us of the unbounded enthusiasm for XML, in the sense that a lot of applications which are currently built based on linked data technologies could be better and more cheaply realized with a classic relational database. At the end of this chapter, the reader hopefully has acquired a sufficient historical, conceptual and technical understanding of which data

Table 2.3 Summary of the (dis)advantages of different data models

data model	(dis-)advantages	use
tabular data	+ intuitive approach + very portable + technology agnostic – prone to redundancy and leading to inconsistencies – inefficient search and retrieval	import and export of data with a simple structure
relational model	+ handling of complex data + optimized queries + mature software market – binary format – schema-dependent	management of complex data which require normalization
meta-markup	+ platform-independent + both human- and machine-readable – complicated implementation for complex data – verbosity	import and export of complex data
RDF	+ schema-neutral approach + discovery of new knowledge – loss of normalization – immature software market	making data available for linking

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

model to use in which context.

To conclude this chapter, Table 2.3 summarizes the essential characteristics of every data model we discussed.

The case study at the end of this chapter will now demonstrate how the linked data approach results in a dynamic view of data, allowing heterogeneous information sources to interconnect in a standardized manner. Through the example queries, issues regarding data inconsistency and incompleteness, which is the drawback of this *open world* approach, will also be underlined. While practising with concrete examples with SPARQL queries, try to reflect specifically on one of the characteristics of the RDF data model that we identified: it is schema-neutral from a conceptual point of view. However, what happens in practice when you want to query a dataset you are not familiar with in SPARQL? We will come back to this issue in the concluding chapter.

7 CASE STUDY: linked data at your fingertips

This case study is slightly different from the others in this book, in that it doesn't focus on a particular dataset. Instead, we will explore linked data from various sources to get a feeling of the possibilities and limitations in practice. Earlier on in this chapter, we referred to the schema-neutral character of the conceptual RDF data model. The exercises of this case study will help you understand how this theoretical model has been implemented in practice. As you will see, the open-world assumption definitively offers opportunities but can be challenging to implement. To demonstrate this, we will retrieve in this case study metadata on Pablo Picasso in various ways, to understand the capabilities of each data source. First, we will examine DBpedia, which is a version of Wikipedia automatically converted into RDF. Next, we will try queries on Freebase, which is a collaborative linked data source with partly automated input (from Wikipedia and other sources) and partly human input. The difference between DBpedia and Freebase is that Freebase can be edited publicly, whereas DBpedia only has a single automated process. Finally, we will zoom in on Sindice, an index that brings together many large datasets of the linked data cloud and is easily accessible through its front-end Sig.ma.

7.1 DBpedia, the Wikipedia of data

DBpedia is a publicly accessible RDF store with content that is automatically extracted from Wikipedia, the free online encyclopedia. A substantial proportion of articles on Wikipedia have semi-structured data in the form of infoboxes (usually displayed to the right of an article) listing key/value data such as names, birth dates, etc., which are available on DBpedia. Two versions of DBpedia are available: a periodically updated version at http://dbpedia.org/ and a *live* version

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

at http://live.dbpedia.org/, which follows the rapid change on Wikipedia. DBpedia currently contains more than 250 million triples. In this case study, we will look at DBpedia and investigate how we can browse and query data.

7.1.1 Browsing DBpedia

Finding a topic page is as easy as going to http://dbpedia.org/page/Topic_Name. For instance, the DBpedia page of Pablo Picasso can be found at http://dbpedia.org/page/Pablo_Picasso. At the top of this page, we see its title Pablo Picasso, followed by a short English description. The remainder of the page consists of a long two-column table with properties and values that contain the information we are interested in. Take your time to look around and discover what DBpedia has to say about Picasso. In addition to human-readable abstracts in many different languages, we see many key-value pairs that contain machine-interpretable information. Using the DBpedia ontology, knowledge about Picasso is expressed, such as birth date and place, multi-language labels, influences, spouse, and nationality.

If you wonder where the triples are, well, you can reconstruct them by taking the page's subject, a predicate from the **Property** column and an object from the **Value** column. For instance, one of the triples is:

```
dbpedia:Pablo_Picasso dbpedia-owl:birthPlace dbpedia:Málaga.
```

There are also several properties in the reverse direction as well, indicated by the key 'is *property* of'. For instance 'is `dbpedia-owl:parent` of' translates to:

```
dbpedia:Paloma_Picasso dbpedia-owl:parent dbpedia:Pablo_Picasso.
```

As we expect from *linked* data, we can click through on any value to learn more. If we click on `dbpedia:Málaga`, we see a page with detailed information on the city. As we explained earlier, even the properties can be examined. Clicking `dbpedia-owl:birthPlace` reveals that this is a relation between a person and a place. Interestingly, some links go outside DBpedia, thus connecting this dataset to others, something that is not possible with relational databases. For instance, `http://data.nytimes.com/N855344257183137093` is indicated as the same resource, with the `owl:sameAs` relation, and this link leads to Picasso's data sheet on the *New York Times* website. This reveals the true potential of linked data.

At the bottom of the page, there are links to view the data in different formats. The 'N3/Turtle' link leads to an RDF serialization that can be interpreted by software. You might notice a lot of strange-looking sequences in the file which take up a large amount of space. They are escape sequences for non-ASCII characters, such as å, ä, or ö for example, from the full-text abstraction fields.

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

However, towards the bottom of the file, you will notice more familiar triples such as:

```
dbpedia:Noel_Rockmore dbpedia-owl:influencedBy
  dbpedia:Pablo_Picasso.
dbpedia:Ben_Shahn dbpedia-owl:influencedBydbpedia:Pablo_Picasso.
dbpedia:Piet_Mondrian dbpedia-owl:influencedBy
  dbpedia:Pablo_Picasso.
```

If possible, we recommend that you switch off line wrapping in your editor, so the long lines with escape sequences will simply flow off the screen.

7.1.2 Querying DBpedia

Once we get to know some basic properties of the data, we have sufficient information to start querying it in a more complex way. We have seen some predicates and some objects, which can help us construct queries. We advise you to keep the Picasso page open in one tab while you bring up the SPARQL query interface at http://dbpedia.org/sparql. You are greeted by the following default query:

```
SELECT DISTINCT ?Concept WHERE { [] a ?Concept. } LIMIT 100
```

We are already familiar with the `WHERE` and `SELECT` clauses, and as their names suggest, `DISTINCT` asks for unique items and `LIMIT 100` for only the first 100 results. The `[]` syntax is a way to say 'any node', like a variable without a name. If we execute this query, we will receive a (quite random) list of 100 Concepts in DBpedia. These are not only DBpedia topics, but also concepts such as `owl:Thing` and `http://schema.org/CreativeWork`.

Let's now try a query of our own, to verify if we can get the same information on Picasso as we did when browsing DBpedia. To see all triples on Picasso, enter the following SPARQL query:[3]

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
SELECT ?p ?o WHERE { dbpedia:Pablo_Picasso ?p ?o. }
```

This will show us all triples we saw earlier on the Picasso page. Well, at least those triples that have Picasso in the subject. To find all triples where Picasso is the object, issue the query:

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

```
SELECT ?s ?p WHERE { ?s ?p dbpedia:Pablo_Picasso. }
```

This yields results like the following:

s	p
dbpedia:The_Three_Dancers	dbpedia-owl:artist
dbpedia:The_Accordionist	dbpedia-owl:artist
dbpedia:Desire_Caught_by_the_Tail	dbpedia-owl:author
dbpedia:Olga_Khokhlova	dbpedia-owl:spouse
dbpedia:Stanley_William_Hayter	dbpedia-owl:influenced
…	…

The values in the s column correspond to bindings of the ?s variable, the p column to bindings of the ?p variable. To understand where the found information comes from, we substitute s and p in the original WHERE clause. The first row thus belongs to a match of:

```
dbpedia:The_Three_Dancers dbpedia-owl:artist dbpedia:Pablo_Picasso.
```

To receive just 30 triples, add the LIMIT clause:

```
SELECT ?p ?o WHERE { dbpedia:Pablo_Picasso ?p ?o. } LIMIT 30
```

If you want to see all predicates used with Picasso as the subject:

```
SELECT ?p WHERE { dbpedia:Pablo_Picasso ?p ?o. }
```

Note the omission of the ?o variable in the SELECT clause, as we only want to see the predicates. This yields the following list:

p
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
…
http://www.w3.org/2002/07/owl#sameAs
http://www.w3.org/2002/07/owl#sameAs
http://www.w3.org/2002/07/owl#sameAs
…
http://purl.org/dc/terms/subject
http://purl.org/dc/terms/subject
http://purl.org/dc/terms/subject
…

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

You might be surprised to see that there are duplicates in the result list. How come we have duplicates here when we did not have them in the previous query? The answer is that triples are unique in the triple store, i.e., a triple can only occur once.[4] However, many triples can use the same predicates, and indeed, several Picasso triples use the `rdf:type` and `owl:sameAs` predicates. To have the unique predicates, we need to add the `DISTINCT` modifier:

```
SELECT DISTINCT ?p WHERE { dbpedia:Pablo_Picasso ?p ?o. }
```

So far, we have received tables of variable values as a result, but what if we want triples? Besides `SELECT`, SPARQL also has a `CONSTRUCT` clause that creates triples instead of variable bindings. For example, this shows all Picasso triples:

```
CONSTRUCT { dbpedia:Pablo_Picasso ?p ?o. }
WHERE     { dbpedia:Pablo_Picasso ?p ?o. }
```

These include the following:

```
dbpedia:Pablo_Picasso rdf:typefoaf:Person,
     yago:SpanishPotters,
     yago:PeopleFromParis,
     yago:BalletDesigners.
dbpedia:Pablo_Picasso dcterms:subject category:Cubism
     category:Spanish_expatriates_in_France,
     category:Spanish_sculptors,
     category:Modern_painters.
```

It might seem strange to duplicate the graph from the `WHERE` clause into the `CONSTRUCT` clause, but they actually signify two different things. The `WHERE` clause tells the SPARQL engine to look for all triples that have Picasso as the subject and to store their predicates and objects in the variables `?p` and `?o` respectively. The `CONSTRUCT` clause instructs the engine to collect all `?p` and `?o` values – regardless of how they were retrieved – and to create triples from them using the specified pattern.

This means that we can choose to generate a different pattern. For instance, suppose that we just want to express that Picasso is somehow connected to the object of the triple, instead of exactly specifying this relationship. Then we can simply do:

```
CONSTRUCT { dbpedia:Pablo_Picasso <isConnectedTo> ?o. }
WHERE     { dbpedia:Pablo_Picasso ?p ?o. }
```

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

This will yield triples such as:

```
dbpedia:Pablo_Picasso <isConnectedTo> category:Modern_painters.
```

This allows you to convert the queried data into the form that you prefer. Finally, to receive 100 random triples from DBpedia, try the following:

```
CONSTRUCT { ?s ?p ?o. } WHERE { ?s ?p ?o. } LIMIT 100
```

7.2 More complex SPARQL queries

`WHERE` patterns can be as complex as you like. The most simple case is a single triple. For instance, the birth place of Picasso can be retrieved by:

```
SELECT ?place WHERE {
  dbpedia:Pablo_Picasso dbpedia-owl:birthPlace ?place.
}
```

This turns out to be http://dbpedia.org/resource/M%C3%A1laga. Note the use of escape sequences in the URL to encode the accented character in 'Málaga'. We can now find all people born in Málaga:

```
SELECT ?person WHERE {
  ?person dbpedia-owl:birthPlace
<http://dbpedia.org/resource/M%C3%A1laga>.
}
```

Unsurprisingly, this list includes Picasso himself:

person
…
dbpedia:Pepe_Romero
dbpedia:Pablo_Picasso
dbpedia:Edu_Ramos
dbpedia:Carlos_Aranda
…

We could simplify the same question by describing the pattern in one query with a `WHERE` clause consisting of two triples:

```
SELECT ?person WHERE {
  dbpedia:Pablo_Picasso dbpedia-owl:birthPlace ?place.
  ?person dbpedia-owl:birthPlace ?place.
}
```

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

This will select all people whose birthplace is the same as Picasso's, without having to specify that exact place. We just instruct the SPARQL engine to find the birthplace for Picasso and look for people with this birthplace at the same time. Were you surprised to see Picasso in the result list? It might seem strange, but this is actually logical: Picasso has the same birthplace as Picasso, hence he is included in the list. Always remember that computers execute what you ask for, not what you intended to ask: Picasso satisfies the query pattern, so his name is returned, even though you already knew this.

We could place further restrictions on this list. For instance, we see many different kinds of people. If we are only interested in *artists* born in Málaga, we can say:

```
SELECT ?person WHERE {
  dbpedia:Pablo_Picasso dbpedia-owl:birthPlace ?place.
  ?person dbpedia-owl:birthPlace ?place.
  ?person a dbpedia-owl:Artist.
}
```

The list has fewer members, and Picasso is still in there (since he's an artist):

person
…
dbpedia:Javier_Conde
dbpedia:Pepe_Romero
dbpedia:Pablo_Picasso
dbpedia:Juan_Antonio_Arguelles_Rius
…

Let's ask for people who were influenced by Picasso:

```
SELECT ?artist WHERE {
  ?artist dbpedia-owl:influencedBy dbpedia:Pablo_Picasso.
  ?artist a dbpedia-owl:Artist.
}
```

And let's see where those people were born, to have an idea of how Picasso's legacy spread geographically:

```
SELECT ?artist, ?place WHERE {
  ?artist dbpedia-owl:influencedBy dbpedia:Pablo_Picasso.
  ?artist a dbpedia-owl:Artist.
  ?artist dbpedia-owl:birthPlace ?place.
}
```

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

This reveals the following data:

artist	place
dbpedia:Sarah_Stein	dbpedia:San_Francisco
dbpedia:Helmut_Kolle	dbpedia:Charlottenburg
dbpedia:Karel_Appel	dbpedia:Amsterdam
dbpedia:Karel_Appel	dbpedia:Netherlands
…	…

Note how some artists occur twice in the list, but with a different place. For instance, Karel Appel has a `birthPlace` of Amsterdam but also The Netherlands, both of which are correct. It might be confusing that they appear twice, but this is because the values are coming from different triples and both answers conform to the query pattern we gave. When we ask for the data as triples, this connection becomes more obvious.

```
CONSTRUCT { ?artist dbpedia-owl:birthPlace ?place. }
WHERE {
  ?artist dbpedia-owl:influencedBy dbpedia:Pablo_Picasso.
  ?artist a dbpedia-owl:Artist.
  ?artist dbpedia-owl:birthPlace ?place.
}
```

Triples indeed better illustrate the connection between the pieces of data:

```
dbpedia:Dick_Bruna dbpedia-owl:birthPlace dbpedia:Utrecht,
                                          dbpedia:Netherlands.
dbpedia:Wifredo_Lam dbpedia-owl:birthPlace dbpedia:Sagua_La_Grande,
                                          dbpedia:Cuba.
dbpedia:Karel_Appel dbpedia-owl:birthPlace dbpedia:Netherlands,
                                          dbpedia:Amsterdam.
```

Another perspective is time: when were the people who were influenced by Picasso born?

```
SELECT ?artist, ?date WHERE {
  ?artist dbpedia-owl:influencedBy dbpedia:Pablo_Picasso.
  ?artist a dbpedia-owl:Artist.
  ?artist dbpedia-owl:birthDate ?date.
}
```

This gives the following people:

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

artist	date
dbpedia:Wifredo_Lam	1902-12-08
dbpedia:Karel_Appel	1921-04-25
dbpedia:Piet_Mondrian	1872-03-07
…	…

We have a considerably shorter list then when we asked for all people influenced by Picasso. This appears strange, because the amount of people should be the same, no matter when they were born. The issue here is that DBpedia does not contain birth data information for all people. As a result, only people with a birth date are included. In contrast, a relational database would include all people but leave unknown birth dates empty. We can obtain the same behaviour from DBpedia by marking the triple with the birth date OPTIONAL:

```
SELECT ?artist, ?date WHERE {
  ?artist dbpedia-owl:influencedBy dbpedia:Pablo_Picasso.
  ?artist a dbpedia-owl:Artist.
  OPTIONAL {
    ?artist dbpedia-owl:birthDate ?date.
  }
}
```

This will give us all influenced people, some of which have an unknown birth date:

artist	date
dbpedia:Wifredo_Lam	1902-12-08
dbpedia:Karel_Appel	1921-04-25
dbpedia:Dick_Bruna	
dbpedia:Piet_Mondrian	1872-03-07
dbpedia:Joan_Glass	
…	…

To obtain a chronological overview, we can ask DBpedia to ORDER the results:

```
SELECT ?artist, ?date WHERE {
  ?artist dbpedia-owl:influencedBy dbpedia:Pablo_Picasso.
  ?artist a dbpedia-owl:Artist.
  OPTIONAL {
    ?artist dbpedia-owl:birthDate ?date.
}
}
ORDER BY ?date
```

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

This will list artists without a known birth date first, followed by an ordered list of those with a known birth date.

Finally, we might be interested to dig even deeper. All artists in the current list were influenced by Picasso, but who did those artists influence?

```
SELECT ?artist, ?influencedArtist WHERE {
  ?artist dbpedia-owl:influencedBy dbpedia:Pablo_Picasso.
  ?influencedArtist dbpedia-owl:influencedBy ?artist.
}
```

The SPARQL engine first looks for all `?artist` matches who were influenced by Picasso. For each `?artist` value, it looks for `?influencedArtist` matches who were influenced by `?artist`:

artist	influencedArtist
dbpedia:Karel_Appel	dbpedia:Jan-Hein_Arens
dbpedia:Piet_Mondrian	dbpedia:Charmion_Von_Wiegand
dbpedia:Piet_Mondrian	dbpedia:Robert_Cottingham
dbpedia:Georges_Braque	dbpedia:Piet_Mondrian
dbpedia:Georges_Braque	dbpedia:Byron_Galvez
…	…

Now is the time to reflect on what we have achieved. The above query selects people influenced by people who were influenced by Picasso. We sincerely challenge you to try finding this information on Google (do let us know how many searches and clicks you needed).

One important remark though: the returned information by DBpedia is *incomplete* and likely *incorrect*. There are two reasons for incompletion. First, not all data might be there. Some artists might not be in DBpedia, others might not have their influences listed. If DBpedia does not know an artist's influences, that result will simply not be included. Second, SPARQL endpoints are not obliged to return all results. RDF has an *open-world assumption*: just as on the web, we are never sure that all information is there. Hence, the SPARQL engine gives its best effort to find your query results, but without the guarantee that they will be complete. You can try to formulate a query that gives you all the data from DBpedia (it's really easy in fact), but you will notice that you only get a few thousand results. It is not that the other triples are not there; it is just that the SPARQL engine decided it has worked hard enough already. The reason for incorrectness, apart from missing information, is that the information in DBpedia stems from the online encyclopedia Wikipedia, which is edited by volunteers. That information might contain errors, or might be outdated. The SPARQL endpoint of

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

DBpedia Live (http://live.dbpedia.org/sparql) might give you more up-to-date results, but unfortunately, no correctness guarantee either. However, results from a search engine such as Google can also be incomplete and incorrect.

7.3 Freebase, the community-curated database

7.3.1 Browsing Freebase

While DBpedia data stems from Wikipedia, an encyclopedia that can be edited by anyone, DBpedia data itself is not publicly editable. Freebase (http://freebase.com/) is a database of linked data where users can add and edit information directly. That does not mean that Freebase contains only human-added data: many entries are loaded automatically from external sources (such as Wikipedia and Netflix) by specialized software tools. However, human curation is an important aspect of the Freebase philosophy.

Freebase is not based on RDF, but its contents are still considered linked data because a triple-like model is followed. Data is organized by topic, and typed relations to other topics can be added; topics and triples are identified by URLs. In contrast to DBpedia, each piece of data also contains fields detailing by whom and when it was created. Furthermore, facts are grouped by *types*. For instance, people have the type *People*, which contains properties such as *Date of birth*, *Gender*, and *Children*. These properties are not always filled in for all members of the *People* type; they just provide an easy template for editors to encourage them to use the correct properties.

Let's have a look at the page for Pablo Picasso on Freebase. You can visit it directly at https://www.freebase.com/m/060_7 or search through the homepage https://www.freebase.com/ for the artist's name. This page starts with an image and an abstract of the topic, as well as various links to pages on the web about the same topic. In the 'Properties' tab below, we see all data on Picasso, grouped by type that belong to categories. The first type is **Topic** and lists general information such as name, description, website and notability. These properties apply to all topics on Freebase. Scrolling down the page, we note other types such as **Person** and **Visual Artist**, as well as **Literature Subject** and **Film Actor**. It might seem strange to label Picasso as a *film actor*, but this is actually merely a grouping of properties per topic. As we expect from linked data, clicking values leads to the information pages about the subjects. Hovering over types and relationships also reveals their properties.

On the top of the page, we can see different tabs. The default view is the **Properties** view. The **I18n** tab (an abbreviation of 'Internationalization') shows attributes with multi-language fields, such as transliterations of Picasso's name in different languages. The **Keys** tab lists different identifiers of the subject, together with their namespace and who created them. Finally, the **Links** tab shows a tabular overview of all properties of the subject, together with the

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

person who added or last edited them and the modification date. This can also serve as a history of all data about the topic.

Freebase is more human-targeted than DBpedia, as the pages are designed to be browsed and edited easily, whereas DBpedia offers primarily data tables. This is already apparent from Freebase's homepage, which allows you to navigate data in an interest-driven way. Similarly to DBpedia, data about topics is also offered in RDF, although this might not be obvious at first sight. Starting from the regular URL for a topic such as https://www.freebase.com/m/060_7, you can obtain the RDF version by changing the URL to http://rdf.freebase.com/rdf/m.060_7.

7.3.2 Querying Freebase

Freebase does not support SPARQL queries; rather, it uses its own query language, MQL, which is based on JSON. The reason for this is historical; SPARQL was not yet standardized when Freebase came into existence. The query form of Freebase is located at https://www.freebase.com/query.

The following query retrieves all personal data about Pablo Picasso:

```
{
  "name": "Pablo Picasso",
  "type": "/people/person",
  "*": null
}
```

The idea is to pass a *template* to the query engine, and all empty fields will be filled out in reply. The fields `name` and `type` narrow the query down to a single topic. The field * ('all') with value `null` ('a single unknown value') instructs Freebase to select all values that belong to this topic. The result is a JSON document similar to the following:

```
{
"result": {
  "place_of_birth": "Málaga",
  "id": "/en/pablo_picasso",
  "parents": [
    "José Ruiz y Blasco",
    "María Picasso y López"
  ],
  "gender": "Male",
...
```

The results you receive depend on the type you specify. If we want to retrieve

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

properties about Picasso's work as an artist, we set the type accordingly:

```
{
  "name": "Pablo Picasso",
  "type": "/visual_art/visual_artist",
  "*": null
}
```

This then results in the following data:

```
{
  "result": {
    "type": "/visual_art/visual_artist",
    "id": "/en/pablo_picasso",
    "name": "Pablo Picasso",
    "art_forms": [
      "Painting",
      "Sculpture",
      "Ceramic art",
...
```

If we are interested in specific properties, we can specify them in the template. For instance, to only show the artworks by Picasso:

```
{
  "id": "/en/pablo_picasso",
  "type": "/visual_art/visual_artist",
  "artworks": []
}
```

The two square brackets `[]` denote an empty list ('multiple unknown values'), which will be filled out by the query engine:

```
{
  "result": {
    "id": "/en/pablo_picasso",
    "type": "/visual_art/visual_artist",
    "artworks": [
      "Guernica",
      "Garçon à la pipe",
      "Les Demoiselles d'Avignon",
...
```

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

Note that a `null` value would not have worked here, because there is more than one artwork.

If we want more details on the artworks, we have to extend the template. For instance, to retrieve all artwork properties:

```
{
  "id": "/en/pablo_picasso",
  "type": "/visual_art/visual_artist",
  "artworks": [{ "*": null }]
}
```

Inside the list, we place a single template object whose properties will be filled in. This returns an extensive list of all works and their properties. If we are only interested in specific properties, we can name them as we did before:

```
{
  "id": "/en/pablo_picasso",
  "type": "/visual_art/visual_artist",
  "artworks": [{ "name": null, "date_completed": null }]
}
```

The templating mechanism thus works on every label of the data. The following query lists all of Picasso's artworks with title and date:

```
{
"result": {
  "artworks": [
    {
      "name": "Guernica",
      "date_completed": "1937-06"
    },
    {
      "name": "Garçon à la pipe",
      "date_completed": "1905"
    },
    {
      "name": "Les Demoiselles d'Avignon",
      "date_completed": "1907"
    },
...
```

The main difference between MQL and SPARQL is that SPARQL still employs

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

the triple model (with variables for unknowns), whereas MQL has proprietary templating mechanisms.

7.4 Sindice, the semantic web index

7.4.1 Querying the entire web

So far, we have only been querying single datasets. However, the central idea of linked data is of course to be able to cross dataset boundaries. On the web, we mostly discover information through search engines, so analogously, a search engine for linked data could help us find datasets and navigate to other datasets from there.

Sindice (http://sindice.com) is an index of the semantic web. It collects linked data in RDF and other formats, and helps you discover more resources. For instance, we can inspect the data Sindice has about Picasso by putting Picasso's URL, http://dbpedia.org/resource/Pablo_Picasso, into the text box and clicking 'Search'. Even though Sindice finds hundreds of matching documents, you might initially be disappointed by the results. While the ranking of results on traditional search engines has considerably improved over the past decades, raking of semantic data is still in its infancy. However, going through the result pages, we discover various datasets that indeed mention Pablo Picasso.

More targeted searches are possible through Sindice's SPARQL endpoint at http://sparql.sindice.com/. For instance, we can see what data Sindice has available on Picasso:

```
SELECT ?s, ?p WHERE { ?s ?p dbpedia:Pablo_Picasso. }
```

In addition to a lot of data from DBpedia, we also see triples from other datasets such as the *New York Times*. The Sindice SPARQL endpoint is also still under development, so not all triples that are available through the search function can be found in the SPARQL endpoint. Another reason we do not see more data is the issue of identity. Indeed, triples in DBpedia are mostly of this form:

```
dbpedia:Pablo_Picasso dbpedia-owl:birthPlace dbpedia:Málaga.
dbpedia:Pablo_Picasso rdf:type dbpedia-owl:Artist.
```

However, in the *New York Times* dataset, triples use another URL to represent Picasso:

```
nytd:N855344257183137093 skos:prefLabel "Picasso, Pablo".
```

This is a common practice in linked data, since there is no central identity

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

authority. Fortunately, both identifiers are connected together by an `owl:sameAs` predicate:

```
nytd:N85534425718313709З owl:sameAs dbpedia:Pablo_Picasso.
```

Therefore, we can find more triples about Picasso by adapting our query. Instead of demanding that the triple contain the DBpedia identifier `dbpedia:Pablo_Picasso`, we say that they can use any identifier, as long as it corresponds to the same concept as `dbpedia:Pablo_Picasso`. In SPARQL, we can express this as:

```
SELECT ?picasso, ?p, ?o
WHERE {
  ?picasso owl:sameAs dbpedia:Pablo_Picasso.
  ?picasso ?p ?o.
}
```

This instructs Sindice to find all `?picasso` identifiers that have a `owl:sameAs` relation to DBpedia's Picasso entry, and for each of them, find all matching triples. We now retrieve results from the *New York Times*, Yago, and several other datasets. This provides an interesting opportunity to harmonize the triples using one single identifier. Therefore, we can construct the triples while explicitly using the DBpedia identifiers as the subject:

```
CONSTRUCT { dbpedia:Pablo_Picasso ?p ?o. }
WHERE {
  ?picasso owl:sameAs dbpedia:Pablo_Picasso.
  ?picasso ?p ?o.
}
```

Note the explicit mention of `dbpedia:Pablo_Picasso` in the `CONSTRUCT` clause, which ensures that all found triples use this identifier, no matter what dataset they originate from.

7.4.2 Browsing Sindice through Sig.ma

While Sindice gives you a more raw view on the web's data, Sig.ma (http://sig.ma/) is an interface built on top of Sindice that lets you collect information about a topic easily. Sig.ma looks through different data sources using Sindice, but groups the information visually together, turning it into a *mash-up*.

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

Figure 2.5 A search for http://dbpedia.org/resource/Pablo_Picasso on Sig.ma

Figure 2.5 shows the result of searching for http://dbpedia.org/resource/Pablo_Picasso. On the left are the different pieces of information, grouped together by category, starting with images at the top. On the right, we see the sources that contributed to the information on the page. They consist of the original URL we entered, but also all URLs that are mentioned on that page. Scrolling down, we see many information sources, and if we hover above them, the corresponding data sources are highlighted on the right. Similarly, we can hover in the sources sidebar on the left to see which pieces of information they contributed.

However, as you would expect, this mostly shows information associated with the single identifier http://dbpedia.org/resource/Pablo_Picasso and so a lot of potential sources are not included. Therefore, we can also search for the artist by typing 'Pablo Picasso' in the search box, which will lead to much more data. However, Sig.ma text search is a little liberal, so some data might only be marginally relevant to Picasso. Even worse, some data is simply incorrect because the topic was not matched precisely. Here are some examples we found:

- Picasso is created by 'darj33ling'. Upon closer inspection of the data source containing this fact, it turns out that this actually refers to a *slide deck* titled 'Pablo Picasso', created by a user account 'darj33ling'.
- Picasso is 425 pixels wide, because a photograph of him is.
- Picasso is a product or service. This seems to come from an online shop that sells puzzles with Picasso's artworks on them.

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

This indicates the kind of problem that can occur when automatically combining data from different sources. It also shows the importance of having URLs as identifiers on the semantic web instead of plain text. Fortunately, Sig.ma allows you to discard sources you do not trust by removing them from the right sidebar. Additionally, you can add new data sources there. Sig.ma will extract data from them on the fly and categorize it on the left.

All in all, Sig.ma is an exciting visualization of the possibilities of linked data, because it connects so many sources together. If we understand its limitations and are careful with selecting sources, very meaningful results can be generated. Do try to put your own name in the search box, as there might even be linked data sources about you that you weren't aware of. And if you have your own data, you can submit it to Sindice so it will be included in Sig.ma results.

Notes

1 We based our overview of data modelling on chapters from Ramsay (2004) and Segaran, Evans and Taylor (2009).

2 See https://en.wikipedia.org/wiki/Law_of_the_instrument.

3 You are not strictly required to add the `PREFIX` declarations here, as DBpedia inserts them automatically for its common prefixes. However, not all SPARQL endpoints support this and so, in general, always include all declarations (even though we will not repeat them in this book due to space constraints).

4 At least, a triple can only occur once in the same graph. Triple stores might contain different graphs, but this is a different story altogether.

References

Berners-Lee, T. (2006) *Linked Data*, http://www.w3.org/DesignIssues/LinkedData.html.

Berners-Lee, T., Hendler, J. and Lassila, O. (2001) The Semantic Web, *Scientific American*, **284** (5), 34–43.

Boiko, B. (2005) *Content Management Bible*, Wiley.

Bray, T., Hollander, D., Layman, A. and Tobin, R. (2006) *Namespaces in XML 1.1*, 2nd edn, W3C Recommendation, http://www.w3.org/TR/xml-names11/.

Lynden, A. and Fenn, J. (2003) *Understanding Gartner's Hype Cycles*, Technical Report, Gartner.

Manovich, L. (2001) *The Language of New Media*, MIT Press.

Ramsay, S. (2004) *A Companion to Digital Humanities*, Blackwell, chapter on Databases.

Segaran, T., Evans, C. and Taylor, J. (2009) *Programming the Semantic Web*, O'Reilly.

Severance, C. (2012) Discovering JavaScript Object Notation, *Computer*, **45** (4), 6–8.

For more Facet books and to order online visit: http://www.facetpublishing.co.uk

Shafranovich, Y. (2005) *Common Format and MIME Type for Comma-separated Values (CSV) Files: request for comments 4180*, Internet Engineering Task Force, http://www.ietf.org/rfc/rfc4180.txt.

For more Facet books and to order online visit: http://www.facetpublishing.co.uk