Resource Description Framework RDF

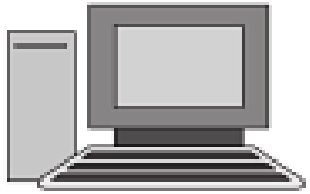
Dr.Khaled ElBahnasy

Tabular Data

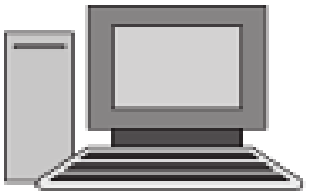
Table 3-1 Tabular Data about Elizabethan Literature and Music

ID	Title	Author	Medium	Year
1	*As You Like It*	Shakespeare	Play	1599
2	*Hamlet*	Shakespeare	Play	1604
3	*Othello*	Shakespeare	Play	1603
4	"Sonnet 78"	Shakespeare	Poem	1609
5	*Astrophil and Stella*	Sir Phillip Sidney	Poem	1590
6	*Edward II*	Christopher Marlowe	Play	1592
7	*Hero and Leander*	Christopher Marlowe	Poem	1593
8	*Greensleeves*	Henry VIII Rex	Song	1525

Distributing Data Across the Web Row by Row

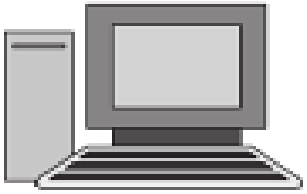


1	*As You Like It*	Shakespeare	Play	1599



Needs common schema—which column is which?

4	Sonnet 78	Shakespeare	Poem	1609
6	Edward II	Christopher Marlowe	Play	1592

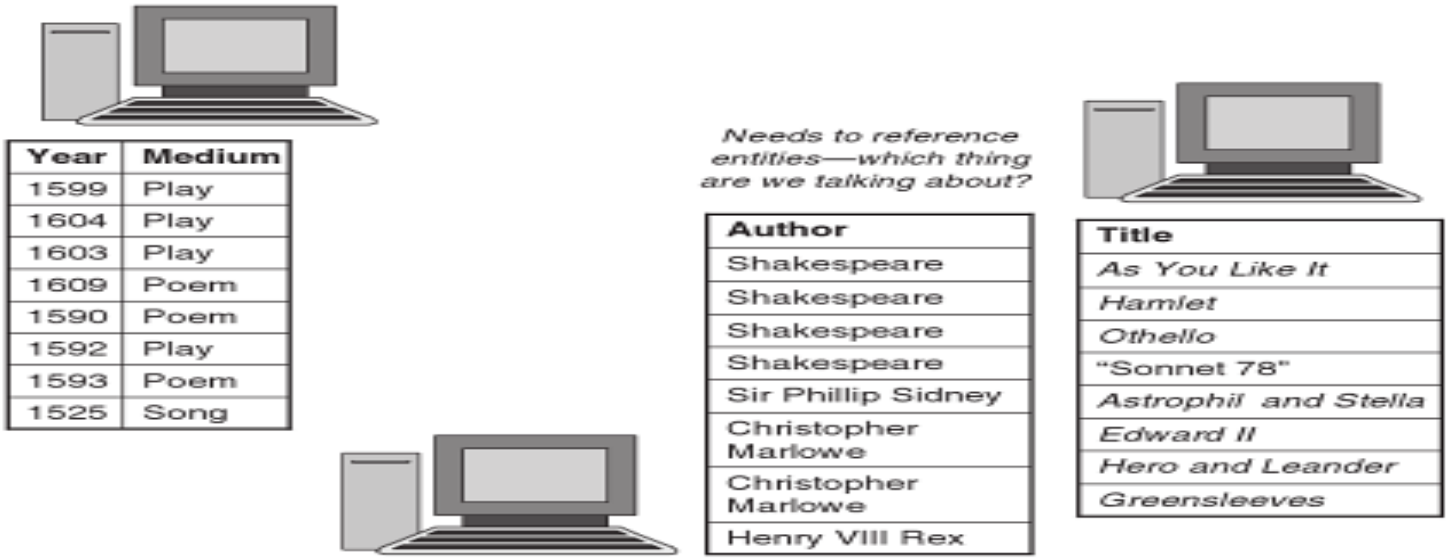


7	Hero and Leander	Christopher Marlowe	Poem	1593
3	Othello	Shakespeare	Play	1603

FIGURE 3-1

Distributing data across the Web, row by row.

Distributing Data Across the Web Column by Column



Year	Medium
1599	Play
1604	Play
1603	Play
1609	Poem
1590	Poem
1592	Play
1593	Poem
1525	Song

Needs to reference entities—which thing are we talking about?

Author
Shakespeare
Shakespeare
Shakespeare
Shakespeare
Sir Phillip Sidney
Christopher Marlowe
Christopher Marlowe
Henry VIII Rex

Title
As You Like It
Hamlet
Othello
"Sonnet 78"
Astrophil and Stella
Edward II
Hero and Leander
Greensleeves

FIGURE 3-2

Distributing data across the Web, column by column.

If we are not interested in the dates of publications, we needn't consider information from that server.

If we want to specify something new about the entities (Number of pages) we can add a new server with that information without disrupting the others.

Distributing Data Across the Web Cell by Cell

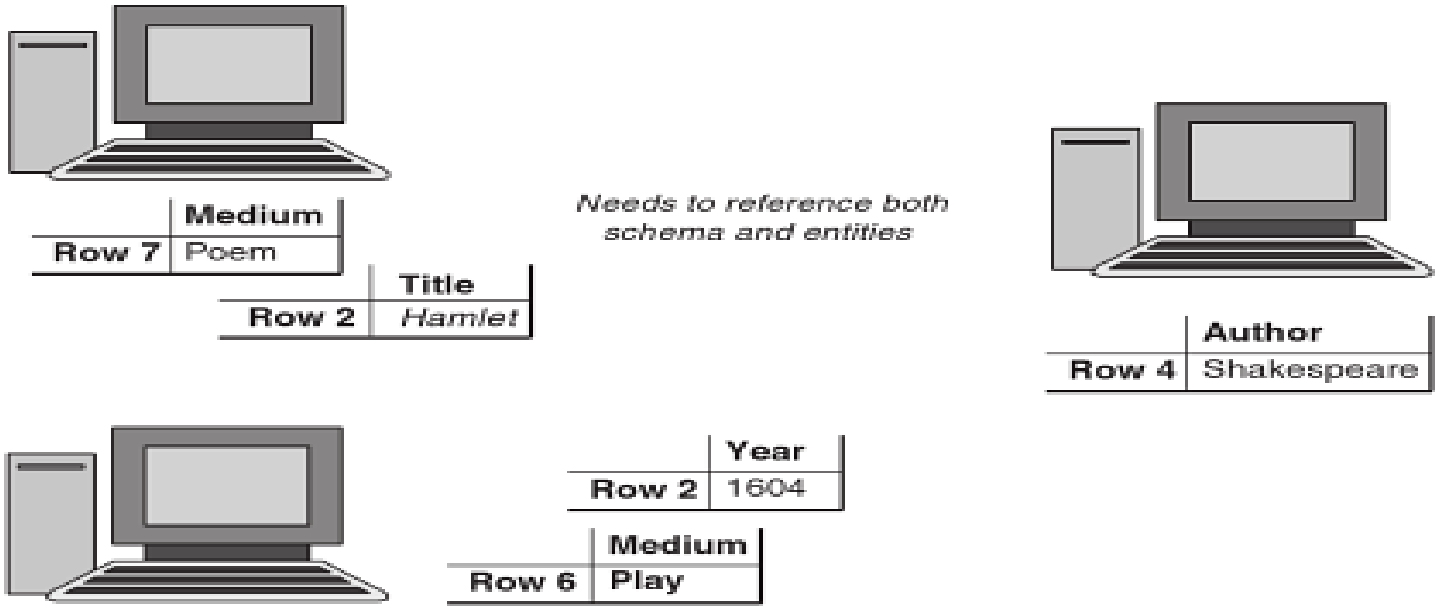


FIGURE 3-3

Distributing data across the Web, cell by cell.

Anyone **A**nything **A**ny topic

triple

Subject	Predicate	Object
Row 7	Medium	Poem
Row 2	Title	Hamlet
Row 2	Year	1604

Table 3.2 Sample Triples

Subject	Predicate	Object
Row 7	Medium	Poem
Row 2	Title	Hamlet
Row 2	Year	1604
Row 4	Author	Shakespeare
Row 6	Medium	Play

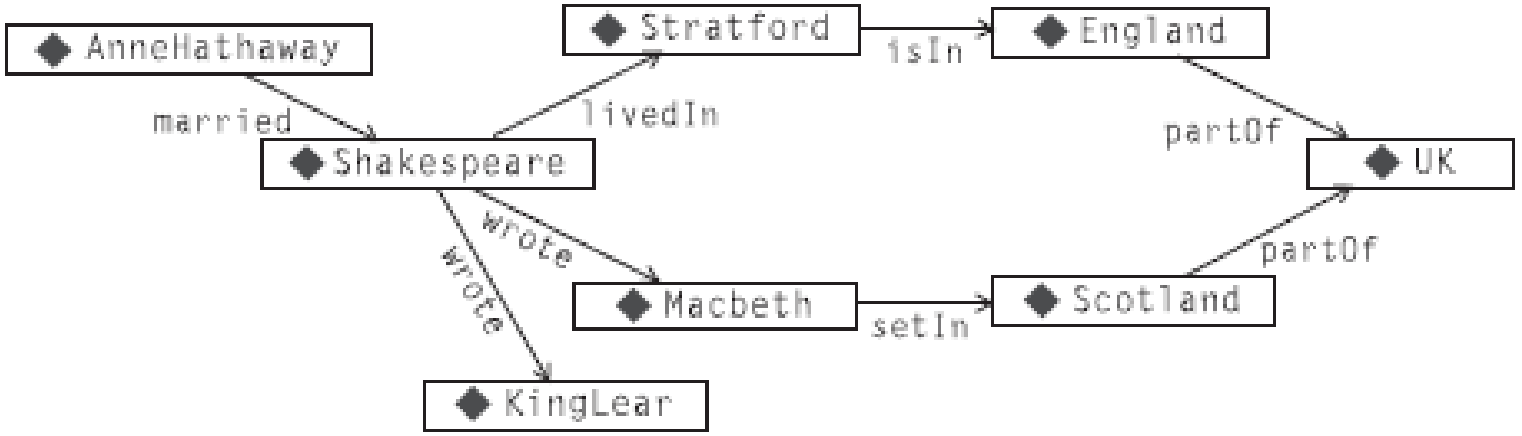
Another example

Subject	Predicate	Object
Shakespeare	Wrote	King Lear
Shakespeare	Lived in	Stratford
Stratford	Is in	England

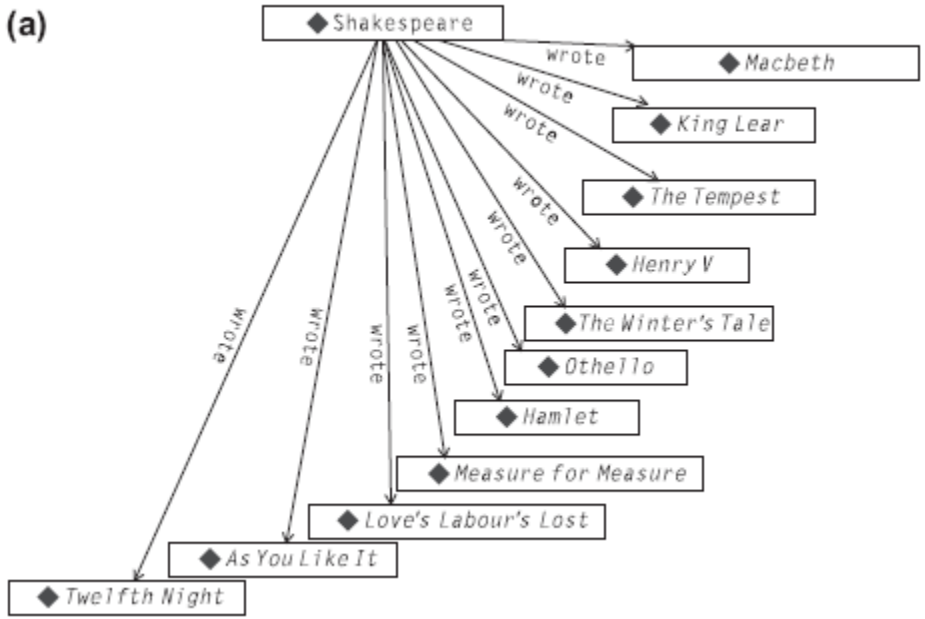
Table 3.3 Sample Triples

Subject	Predicate	Object
Shakespeare	wrote	King Lear
Shakespeare	wrote	Macbeth
Anne Hathaway	married	Shakespeare
Shakespeare	livedIn	Stratford
Stratford	isIn	England
Macbeth	setIn	Scotland
England	partOf	UK
Scotland	partOf	UK

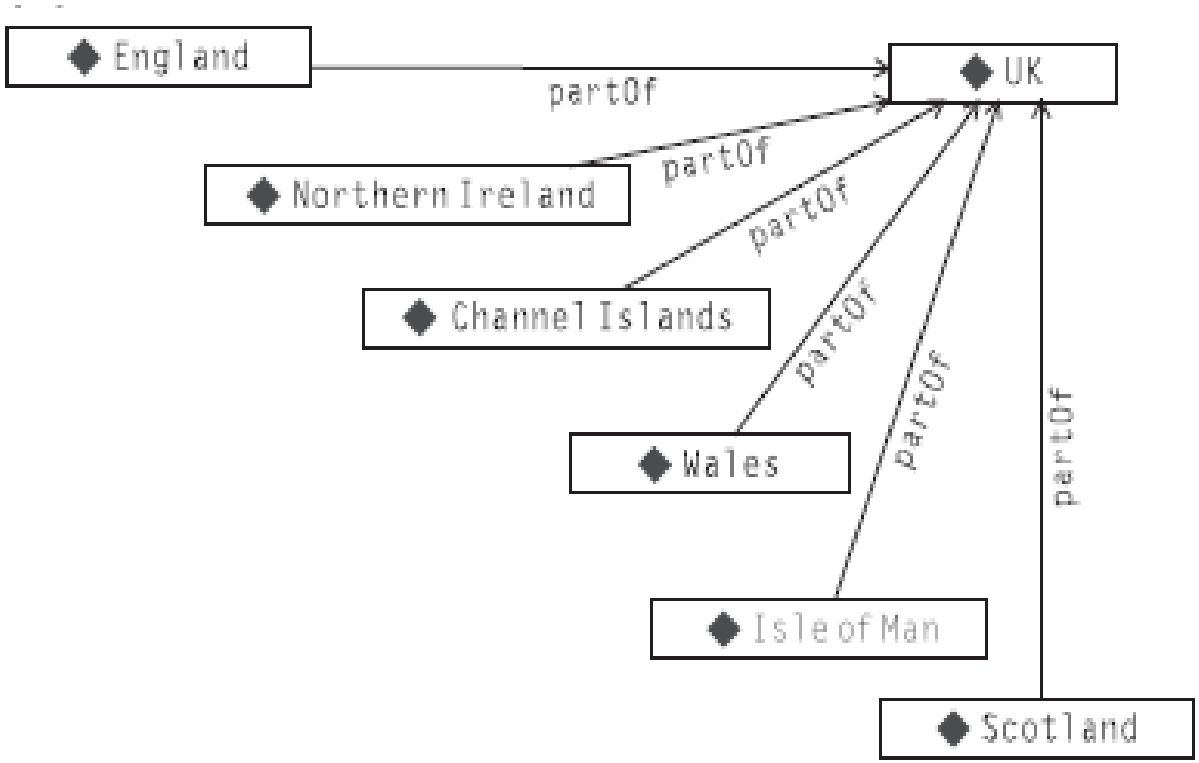
Graph rep.


AnneHathaway
Stratford
isIn
England
married
livedIn
partOf
Shakespeare
UK
wrote
wrote
partOf
Macbeth
setIn
Scotland
KingLear

MERGING DATA FROM MULTIPLE SOURCES

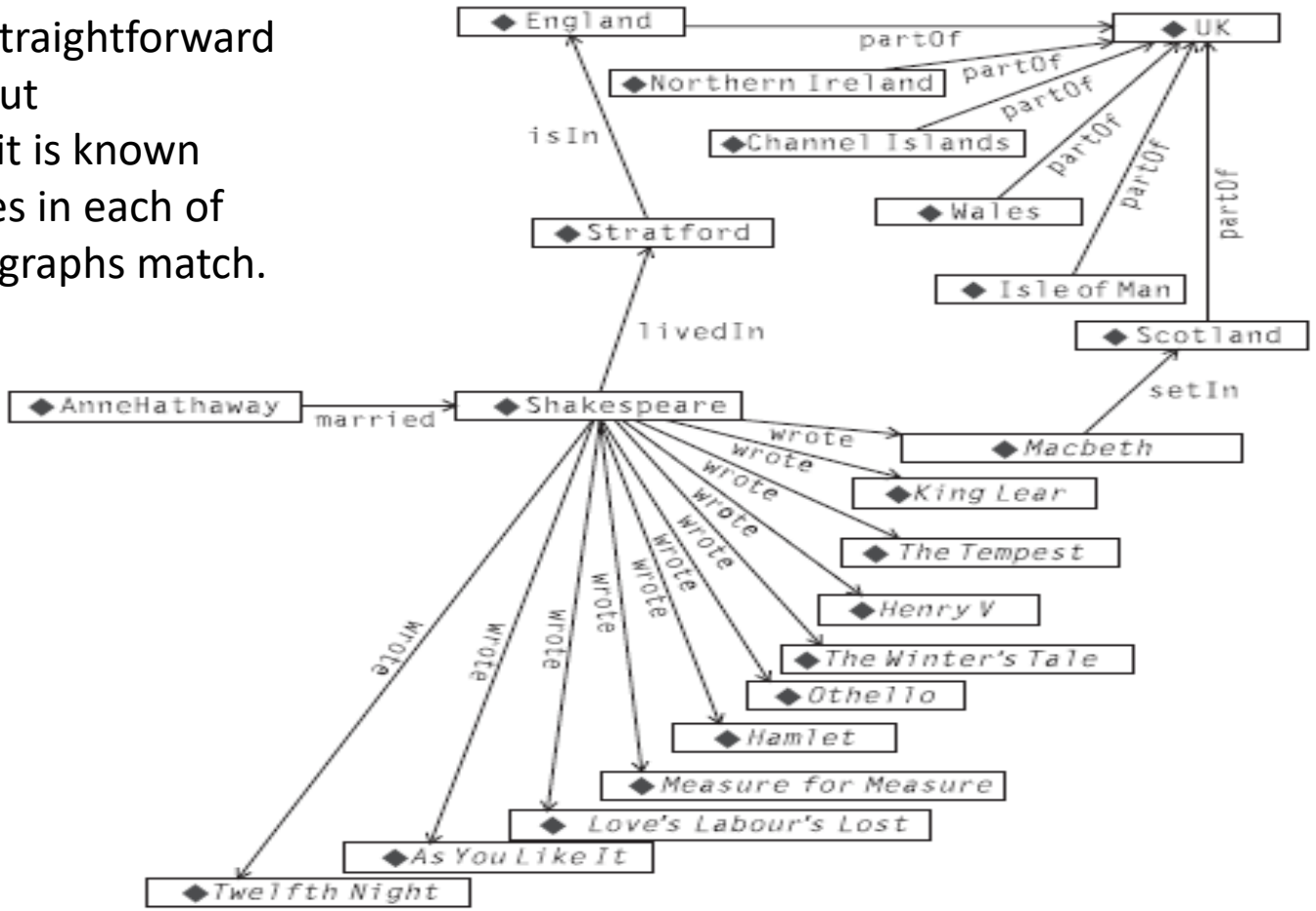


MERGING DATA FROM MULTIPLE SOURCES



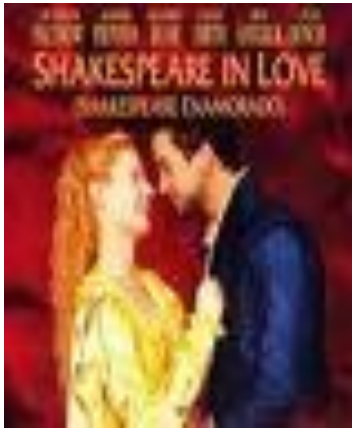
MERGING DATA FROM MULTIPLE SOURCES

create a combined graph like the one shown in the figure is a straightforward process—but only when it is known which nodes in each of the source graphs match.



NAMESPACES, URIS, AND IDENTITY (Cont.)

- The essence of the merge comes down to answering the question "When is a node in one graph the same node as a node in another graph?" In RDF, this issue is resolved through the use of Uniform Resource Identifiers (URIs).
- we have labeled the nodes and edges in the graphs with simple names like Shakespeare or Wales. On the Semantic Web, this is not sufficient information to determine whether two nodes are really the same. Why not?
- The URI (which specifies things like server name, protocol, port number, file name, etc.) to locate a file (or a location in a file) on the Web.
- RDF applies the notion of the URI to resolve the Identity problem in the graph.



NAMESPACES, URIS, AND IDENTITY

- URIs work very well for expressing identity on the World Wide Web, but they are typically a bit of a pain to write out in detail when expressing models, especially in print. So for the examples in this book, we use a simplified version of a URI abbreviation scheme called qnames
- Qnames is a namespace and an identifier written with a colon between.
- The RDF/XML standard includes elaborate rules that allow programmers to map namespaces to other URI representations (such as the familiar http:// notation).
- We follow the InterCap convention (sometimes called CamelCase), whereby names that are made up of multiple words are transformed into identifiers without spaces by capitalizing each word. Thus, “part of ” becomes partOf, “Great Britain” becomes GreatBritain, “Measure for Measure” becomes
- MeasureForMeasure, and so on.

Qnames

Table 3.6 Plays of Shakespeare with Qnames

Subject	Predicate	Object
lit:Shakespeare	lit:wrote	lit:AsYouLikeIt
lit:Shakespeare	lit:wrote	lit:HenryV
lit:Shakespeare	lit:wrote	lit:LovesLaboursLost
lit:Shakespeare	lit:wrote	lit:MeasureForMeasure
lit:Shakespeare	lit:wrote	lit:TwelfthNight
lit:Shakespeare	lit:wrote	lit:WintersTale
lit:Shakespeare	lit:wrote	lit:Hamlet
lit:Shakespeare	lit:wrote	lit:Othello
		etc.

Table 3.7 Geographical Information as Qnames

Subject	Predicate	Object
geo:Scotland	geo:partOf	geo:UK
geo:England	geo:partOf	geo:UK
geo:Wales	geo:partOf	geo:UK
geo:NorthernIreland	geo:partOf	geo:UK
geo:ChannelIslands	geo:partOf	geo:UK
geo:IsleOfMan	geo:partOf	geo:UK

lit stands for http://www.WorkingOntologist.com/Examples/Chapter3/Shakespeare#, and

geo stands for http://www.WorkingOntologist.com/Examples/Chapter3/geography#.

Qnames

Table 3.8 Triples Referring to URIs with a Variety of Namespaces

Subject	Predicate	Object
lit:Shakespeare	lit:wrote	lit:KingLear
lit:Shakespeare	lit:wrote	lit:MacBeth
bio:AnneHathaway	bio:married	lit:Shakespeare
bio:AnneHathaway	bio:livedWith	lit:Shakespeare
lit:Shakespeare	bio:livedIn	geo:Stratford
geo:Stratford	geo:isIn	geo:England
geo:England	geo:partOf	geo:UK
geo:Scotland	geo:partOf	geo:UK

Standard namespaces

- The W3C has defined a number of standard namespaces for use with Web technologies, including xsd: for XML schema definition; xmlns: for XML namespaces; and so on.
- rdf: Indicates identifiers used in RDF. The set of identifiers defined in the standard is quite small and is used to define types and properties in RDF. The global URI for the rdf namespace is http://www.w3.org/1999/02/22-rdf-syntax-ns#.
- rdfs: Indicates identifiers used for the RDF Schema language, RDFS. The scope and semantics of the symbols in this namespace are the topics of future chapters. The global URI for the rdfs namespace is http://www.w3.org/2000/01/rdf-schema#.
- owl: Indicates identifiers used for the Web Ontology Language, OWL. The scope and semantics of the symbols in this namespace are the topics of future chapters. The global URI for the owl namespace is http://www.w3.org/2002/07/owl#.

Standard namespaces

http://www.w3.org/2000/01/rdf-schema#subClassOf,

Or

rdfs:subClassOf for short

refers to a particular term that the W3C makes some statements about in the RDFS standard. But the term can also be dereferenced—that is, if we look at the server www.w3.org, there is a page at the location 2000/01/rdf-schema with an entry about subClassOf, giving supplemental information about this resource

IDENTIFIERS IN THE RDF NAMESPACE

Table 3.9 Using `rdf:type` to Describe Playwrights

Subject	Predicate	Object
lit:Shakespeare	rdf:type	lit:Playwright
lit:Ibsen	rdf:type	lit:Playwright
lit:Simon	rdf:type	lit:Playwright
lit:Miller	rdf:type	lit:Playwright
lit:Marlowe	rdf:type	lit:Playwright
lit:Wilder	rdf:type	lit:Playwright

Table 3.10 Defining Types of Names

Subject	Predicate	Object
lit:Playwright	rdf:type	bus:Profession
bus:Profession	rdf:type	hr:Compensation

Table 3.11 `rdf:Property` Assertions for Tables 3.5 to 3.8

Subject	Predicate	Object
lit:wrote	rdf:type	rdf:Property
geo:partOf	rdf:type	rdf:Property
bio:married	rdf:type	rdf:Property
bio:livedIn	rdf:type	rdf:Property
bio:livedWith	rdf:type	rdf:Property
geo:isIn	rdf:type	rdf:Property

CHALLENGE: RDF AND TABULAR DATA

Table 3.12 Sample Tabular Data for Triples

Product						
ID	**Model Number**	**Division**	**Product Line**	**Manufacture Location**	**SKU**	**Available**
1	ZX-3	Manufacturing support	Paper machine	Sacramento	FB3524	23
2	ZX-3P	Manufacturing support	Paper machine	Sacramento	KD5243	4
3	ZX-3S	Manufacturing support	Paper machine	Sacramento	IL4028	34
4	B-1430	Control engineering	Feedback line	Elizabeth	KS4520	23
5	B-1430X	Control engineering	Feedback line	Elizabeth	CL5934	14
6	B-1431	Control engineering	Active sensor	Seoul	KK3945	0
7	DBB-12	Accessories	Monitor	Hong Kong	ND5520	100
8	SP-1234	Safety	Safety valve	Cleveland	HI4554	4
9	SPX-1234	Safety	Safety valve	Cleveland	OP5333	14

CHALLENGE: RDF AND TABULAR DATA

Table 3.13 Triples Representing Some of the Data in Table 3.12

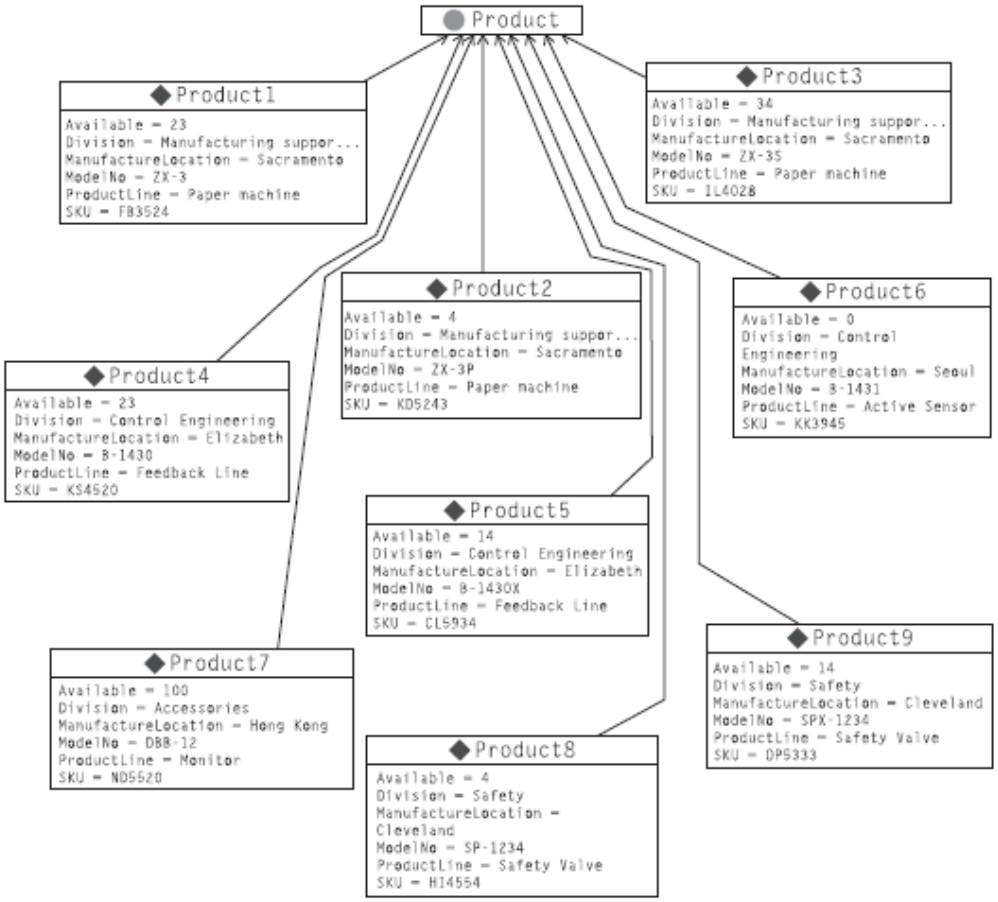
Subject	Predicate	Object
mfg:Product1	mfg:Product_ID	1
mfg:Product1	mfg:Product_ModelNo	ZX-3
mfg:Product1	mfg:Product_Division	Manufacturing support
mfg:Product1	mfg:Product_Product_Line	Paper machine
mfg:Product1	mfg:Product_Manufacture_Location	Sacramento
mfg:Product1	mfg:Product_SKU	FB3524
mfg:Product1	mfg:Product_Available	23
mfg:Product2	mfg:Product_ID	2
mfg:Product2	mfg:Product_ModelNo	ZX-3P
mfg:Product2	mfg:Product_Division	Manufacturing support
mfg:Product2	mfg:Product_Product_Line	Paper machine
mfg:Product2	mfg:Product_Manufacture_Location	Sacramento
mfg:Product2	mfg:Product_SKU	KD5243
mfg:Product2	mfg:Product_Available	4…

CHALLENGE: RDF AND TABULAR DATA

Table 3.14 Triples Representing Type of Information from Table 3.12

Subject	Predicate	Object
mfg:Product1	rdf:type	mfg:Product
mfg:Product2	rdf:type	mfg:Product
mfg:Product3	rdf:type	mfg:Product
mfg:Product4	rdf:type	mfg:Product
mfg:Product5	rdf:type	mfg:Product
mfg:Product6	rdf:type	mfg:Product
mfg:Product7	rdf:type	mfg:Product
mfg:Product8	rdf:type	mfg:Product
mfg:Product9	rdf:type	mfg:Product

CHALLENGE: RDF AND TABULAR DATA



ALTERNATIVES FOR SERIALIZATION

- So far, we have expressed RDF triples in subject/predicate/object tabular form or as graphs of boxes and arrows.
- Although these are simple and apparent forms to display triples, they aren’t always the most compact forms, or even the most human-friendly form, to see the relations between entities.
- The issue of representing RDF in text doesn’t only arise in books and documents about RDF; it also arises when we want to publish data in RDF on the Web.
- In response to this need, there are multiple ways of expressing RDF in textual form.

N-Triples

- The simplest form is called N-Triples and corresponds most directly to the raw RDF triples. It refers to resources using their fully unabbreviated URIs. Each URI is written between angle brackets (< and >).
- Three resources are expressed in subject/predicate/object order, followed by a period (.).
- For example, if the namespace mfg corresponds to http://www.WorkingOntologist.org/Examples/Chapter3Manufacture.rdf#,

  then the first triple from Table 3.14 is written in N-Triples as follows:

<http://www.WorkingOntologist.org/Examples/Chapter3Manufacture.rdf#product1>

<http://www.w3.org/1999/02/22 - syntax-ns#type>

<http://www.WorkingOntologist.org/Examples/Chapter3Manufacture.rdf#Product>

- It is difficult to print N-Triples on a page in a
- An actual ntriple file has the whole triple on a single line.

Turtle

- We use a more compact serialization of RDF called Turtle.
- Turtle combines the apparent display of triples from N-Triples with the terseness of qnames.
- We will introduce Turtle in this section and describe just the subset required for the current examples.
- We will describe more of the language as needed for later examples.
- For a full description of Turtle, see the W3C Turtle team submission
- Since Turtle uses qnames, there must be a binding between the (local) qnames and the (global)URIs. Hence, Turtle begins with a preamble in which these bindings are defined; for example, we can define the qnames needed in the Challenge example with the following preamble:

@prefix mfg:
<http://www.WorkingOntologist.com/Examples/Chapter3/Manufacturing#>
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#

Turtle

- Once the local qnames have been defined, Turtle provides a very simple way to express a triple by listing three resources, using qname abbreviations, in subject/predicate/object order, followed by a period, such as the following:

  mfg:Product1 rdf:type mfg:Product .

- The final period can come directly after the resource for the object, but we often put a space in front of it, to make it stand out visually.
- This space is optional.

Turtle

- It is quite common (especially after importing tabular data) to have several triples that share a common subject.
- Turtle provides for a compact representation of such data.
- It begins with the first triple in subject/predicate/object order, as before; but instead of terminating with a period, it uses a semicolon (;) to indicate that another triple with the same subject follows.
- For that triple, only the predicate and object need to be specified (since it is the same subject from before).

Turtle

- The information in Tables 3.13 and 3.14 about Product1 and Product2 appears in Turtle as follows:
- mfg:Product1 rdf:type mfg:Product;
- mfg:Product_Division “Manufacturing support”;
- mfg:Product_ID “1”;
- mfg:Product_Manufacture_Location “Sacramento”;
- mfg:Product_ModelNo “ZX-3”;
- mfg:Product_Product_Line “Paper Machine”;
- mfg:Product_SKU “FB3524”;
- mfg:Product_Available “23” .
- mfg:Product2 rdf:type mfg:Product;
- mfg:Product_Division “Manufacturing support”;
- mfg:Product_ID “2”;
- mfg:Product_Manufacture_Location “Sacramento”;
- mfg:Product_ModelNo “ZX-3P”;
- mfg:Product_Product_Line “Paper Machine”;
- mfg:Product_SKU “KD5243”;
- mfg:Product_Available “4”

Turtle

- When there are several triples that share both subject and predicate, Turtle provides a compact way to express this as well so that neither the subject nor the predicate needs to be repeated.
- Turtle uses a comma (,) to separate the objects.
- So the fact that Shakespeare had three children named Susanna, Judith, and Hamnet can be expressed as follows:
- lit:Shakespeare b:hasChild b:Susanna, b:Judith, b:Hamnet.
- There are actually three triples represented here—namely:
- lit:Shakespeare b:hasChild b:Susanna.
- lit:Shakespeare b:hasChild b:Judith.
- lit:Shakespeare b:hasChild b:Hamnet.

Turtle

- Turtle provides some abbreviations to improve terseness and readability; in this book, we use just a few of these.
- One of the most widely used abbreviations is to use the word a to mean rdf:type.
- The motivation for this is that in common speech, we are likely to say, “Product1 is a Product” or “Shakespeare is a playwright” for the triples,

  mfg:Product1 rdf:type mfg:Product.

  lit:Shakespeare rdf:type lit:Playwright.
- respectively. Thus we will usually write instead:

  mfg:Product1 a mfg:Product.

  lit:Shakespeare a lit:Playwright.

RDF/XML

- While Turtle is convenient for human consumption and is more compact for the printed page, many Web infrastructures are accustomed to representing information in HTML or, more generally, XML.
- For this reason, the W3C has recommended the use of an XML serialization of RDF called RDF/XML.
- The information about Product1 and Product2 just shown looks as follows in RDF/XML.
- In this example, the subjects (Product1 and Product2) are referenced using the XML attribute rdf:about; the triples with each of these as subjects appear as subelements within these definitions.
- The complete details of the RDF/XML syntax are beyond the scope of this discussion and can be found in http://www.w3.org/TR/rdf-syntax-grammar/.

Example

```
<rdf:RDF
xmlns:mfg="http://www.WorkingOntologist.com/Examples/Chapter3/
Manufacturing#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntaxns#">
<mfg:Product rdf:about="http://www.WorkingOntologist.com/Examples/Chapter3/Manufacturing#Product1">
<mfg:Available>23</mfg:Available>
<mfg:Division>Manufacturing support</mfg:Division>
<mfg:ProductLine>Paper machine</mfg:ProductLine>
<mfg:SKU>FB3524</mfg:SKU>
<mfg:ModelNo>ZX-3</mfg:ModelNo>
<mfg:ManufactureLocation>Sacramento</mfg:Manufacture Location>
</mfg:Product>

<mfg:Product
rdf:about="http://www.WorkingOntologist.com/Examples/Chapter3/
Manufacturing#Product2">
<mfg:SKU>KD5243</mfg:SKU>
<mfg:Division>Manufacturing support</mfg:Division>
<mfg:ManufactureLocation>Sacramento</mfg:Manufacture
Location>
<mfg:Available>4</mfg:Available>
<mfg:ModelNo>ZX-3P</mfg:ModelNo>
<mfg:ProductLine>Paper machine</mfg:ProductLine>
</mfg:Product>
</rdf:RDF>
```

Note

- The same information is contained in the RDF/XML form as in the Turtle, including the declarations of the qnames for mfg: and rdf:.
- RDF/XML includes a number of rules for determining the fully qualified URI of a resource mentioned in an RDF/XML document.
- These details are quite involved and will not be used for the examples in this book.

SPARQL Query Language for RDF

- The SPARQL query language includes a protocol for communicating queries and results so that a query engine can act as a web service.
- This provides another source of data for the semantic web—the so-called SPARQL endpoints provide access to large amounts of structured RDF data.
- It is even possible to provide SPARQL access to databases that are not triple stores, effectively translating SPARQL queries into the query language of the underlying store.
- The W3C has recently begun the process to standardize a translation from SPARQL to SQL for relational stores.

Comparison to relational queries

- In many ways, an RDF query engine is very similar to the query engine in a relational data store: It provides a standard interface to the data and defines a formalism by which data are viewed.
- A relational query language is based on the relational algebra of joins and foreign key references.
- RDF query languages look more like statements in predicate calculus. Unification variables are used to express constraints between the patterns.

Comparison to relational queries

- A relational query describes a new data table that is formed by combining two or more source tables.
- An RDF query (whether in SPARQL or another RDF query language) can describe a new graph that is formed by describing a subset of a source RDF graph.
- That graph, in turn, may be the result of having merged together several other graphs.
- The inherently recursive nature of graphs simplifies a number of detailed issues that arise in table-based queries.
- For instance, an RDF query language like SPARQL has little need for a subquery construct; in many cases, the same effect can be achieved with a single query.
- Similarly, there is nothing corresponding to the special case of an SQL “self-join” in SPARQL.

Comparison to relational queries

- In the special case in which an RDF store is implemented as a single table in a relational database, any graph pattern match in such a scenario will constitute a self-join on that table.
- Some end developers choose to work this way in a familiar SQL environment. Oracle takes another approach to making RDF queries accessible to SQL programmers by providing its own SPARQL extension to its version of SQL, optimized for graph queries.
- Their SPARQL engine is smoothly integrated with the table/join structure of their SQL scripting language.

APPLICATION CODE

- Database applications include more than just a database and query engine; they also include some application code, in an application environment, that performs some analysis on or displays some information from the database.
- The only access the application has to the database is through the query interface, as shown in Figure 4.1.
- An RDF application has a similar architecture, but it includes the RDF parser and serializer, converters, the RDF merge functionality, and the RDF query engine. These capabilities interact with the application itself and the RDF store as shown in Figure 4.2.
- The application itself can take any of several forms. Most commonly, it is written in a conventional
- programming language (Java, C, Python, and Perl are popular options). In this case, the RDF capabilities are provided as API bindings for that language. It is also common for an RDF store to provide a scripting language as part of the query system, which gives programmatic access to these capabilities in a way that is not unlike how advanced dialects of SQL provide scripting capabilities for relational database applications.

APPLICATION CODE

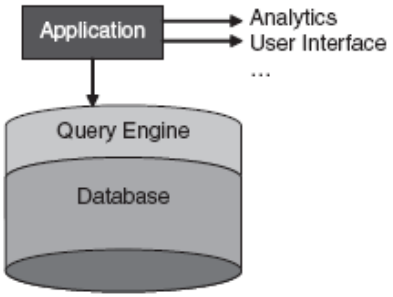


FIGURE 4.1

Application architecture for a database application.

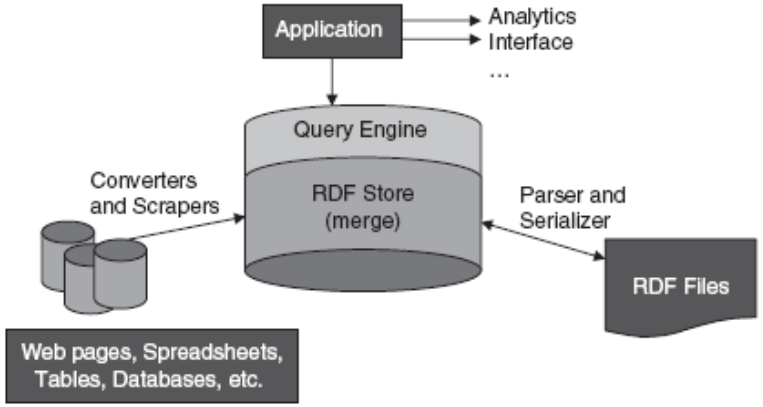


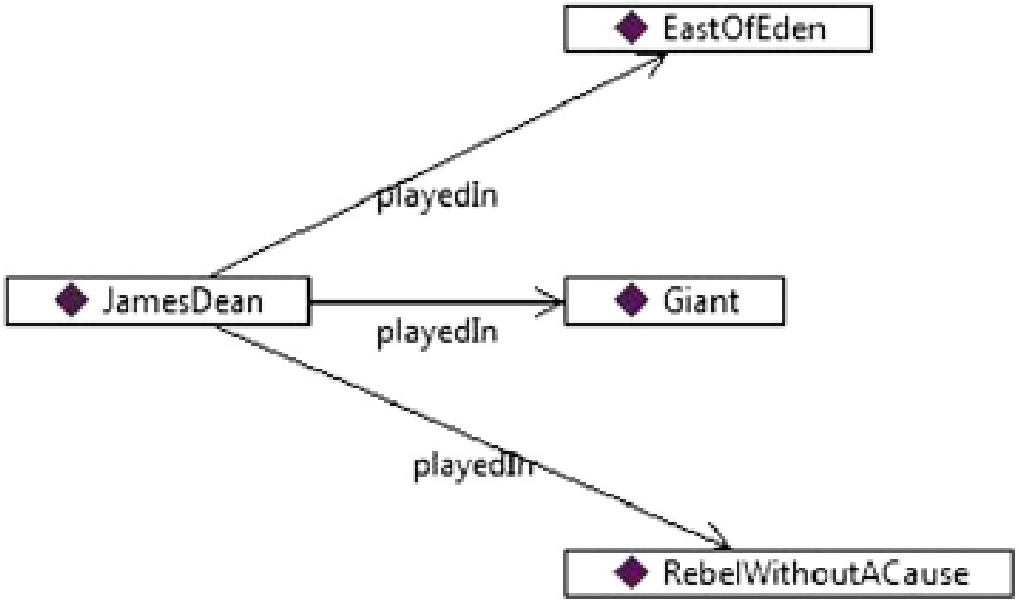
FIGURE 4.2

Application architecture for an RDF application.

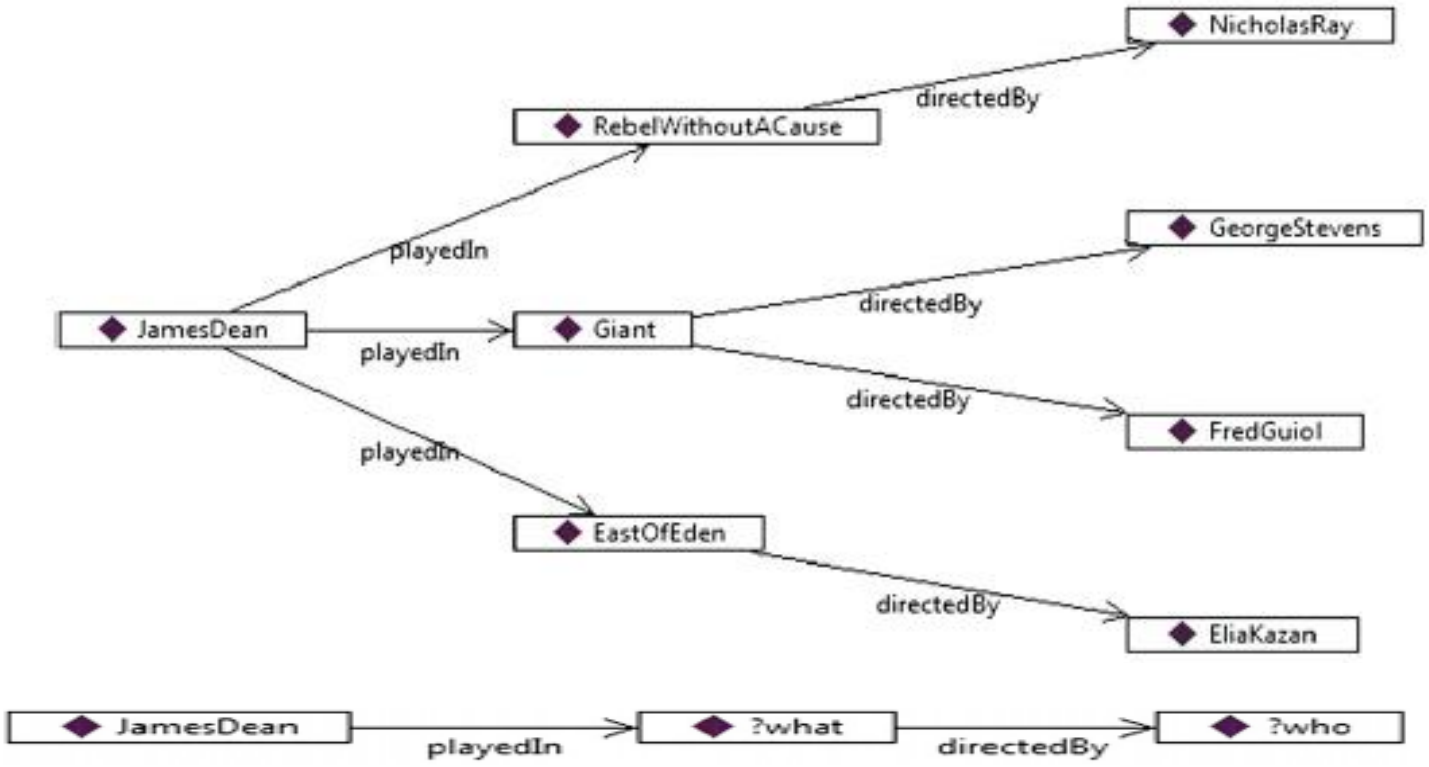
Example

- SPARQL—QUERY LANGUAGE FOR RDF
- The syntax of SPARQL actually looks like Turtle. So these examples really look more like this:
- **Tell: :JamesDean :playedIn :Giant .**
- **Ask: :JamesDean :playedIn ?what .**
- **Answer: :Giant**
- **Ask: ?who :playedIn :Giant .**
- **Answer: :JamesDean**
- **Ask: :JamesDean ?what :Giant .**
- **Answer: :playedIn**

Graph Example



Graph Example



Query Example

- A SPARQL SELECT query has two parts; a set of question words, and a question pattern.
- The keyword WHERE indicates the selection pattern, written in braces. We have already seen some question patterns, e.g.,

- **WHERE {:JamesDean :playedIn ?what .}**
- **WHERE {?who :playedIn :Giant .}**
- **WHERE {:JamesDean ?what :Giant .}**

- The query begins with the word SELECT followed by a list of the question words. So the queries for the questions above are
- **SELECT ?what WHERE {:JamesDean ?playedIn ?what .}**
- **SELECT ?who WHERE {?who :playedIn :Giant .}**
- **SELECT ?what WHERE {:JamesDean ?what :Giant .}**

- It might seem that listing the question words in the SELECT part is redundant—after all, they appear in the patterns already. To some extent, this is true, but we'll see later how modifying this list can be useful.

Query Example

- RDF (and SPARQL) deals well with multiple values. If we TELL the system that James Dean played in multiple movies, we can do this without having to make any considerations in the representation:

Tell: :JamesDean :playedIn :Giant .

Tell: :JamesDean :playedIn :EastOfEden .

Tell: :JamesDean :playedIn :RebelWithoutaCause .

- Now if we ASK a question with SPARQL

Ask: SELECT ?what WHERE {:JamesDean :playedIn ?what}

Answer: :Giant, :EastOfEden, :RebelWith

SIMPLE KNOWLEDGE ORGANIZATION SYSTEM (SKOS)

- SKOS (the Simple Knowledge Organization System) is a W3C Recommendation that provides a means for representing knowledge organization systems (including controlled vocabularies, thesauri, taxonomies, and folk sonomies) in a distributed and linkable way. Knowledge Organization Systems have been around for a long time, most formally as part of Library Science, but means for representing and exchanging them in a computer network have not.
- Given the existence of several thesaurus standards, one could well wonder why people found it necessary to create another one. The key differentiator between SKOS and existing thesaurus standards is its basis in the SemanticWeb. Unlike existing standards, SKOS was designed from the start to allow modelers to create modular knowledge organizations that can be reused and referenced across theWeb.
- SKOS was not designed to replace any thesaurus standard but in fact to augment them by bringing the distributed nature of the SemanticWeb to thesauri and controlled vocabularies.
- Toward this end, it was also a design goal of SKOS that it be possible to map any thesaurus standards to SKOS in a fairly straightforward way.

SKOS

- As an example of using SKOS, for many years, the United Nations Food and Agriculture Organization has maintained a thesaurus called AGROVOC for organizing documents about agriculture.
- Figure 10.1 shows a sample from the SKOS publication of AGROVOC. The diagram shows six concepts, which are related to one another by various properties that are defined in the SKOS Core.
- Data properties are shown within the boxes corresponding to the concepts. As we shall see, each of these properties is defined in relation to other properties, so certain useful inferences can be made.

AGROVOC

FIGURE 10.1

Sample concepts from AGROVOC.

SKOS

- The same information from Figure 10.1 is shown as triples in Turtle here:
- agrovoc:c_4397
  a skos:Concept ;
  skos:prefLabel "Bétail"@fr , "Livestock"@en ;
  skos:altLabel "Animal stock"@en , "Farm animals"@en .
- agrovoc:c_2746
  a skos:Concept ;
  skos:prefLabel "Brebis"@fr , "Ewes"@en , “ماشية”@ar;
  skos:altLabel "Gimmers"@en , "Antenaise"@fr , "Ewe hoggs"@en .
- agrovoc:c_4163
  a skos:Concept ;
  skos:prefLabel "Agneau"@fr , "Lambs"@en .
- agrovoc:c_6443
  a skos:Concept ;
  skos:prefLabel "Bélier"@fr , "Rams"@en ;
  skos:altLabel "Tups"@en .

- agrovoc:c_8369
- a skos:Concept ;
- skos:prefLabel "Wethers"@en , "Mouton"@fr ;
- skos:altLabel "Wether hoggs"@en , "Hoggets"@en ;
- skos:scopeNote "Ovin mâle castré"@fr , "Castratedmalesheep"@en.
- agrovoc:c_7030
- a skos:Concept ;
- skos:prefLabel "Sheep"@en , "Ovin"@fr ;
- skos:altLabel "Ovis aries"@en , "Ovis aries"@fr ;
- skos:broader agrovoc:c_4397 ;
- skos:narrower agrovoc:c_2746, agrovoc:c_6443, agrovoc:c_8369,
- agrovoc:c_4163 .