Data Modeling in a Big Data World

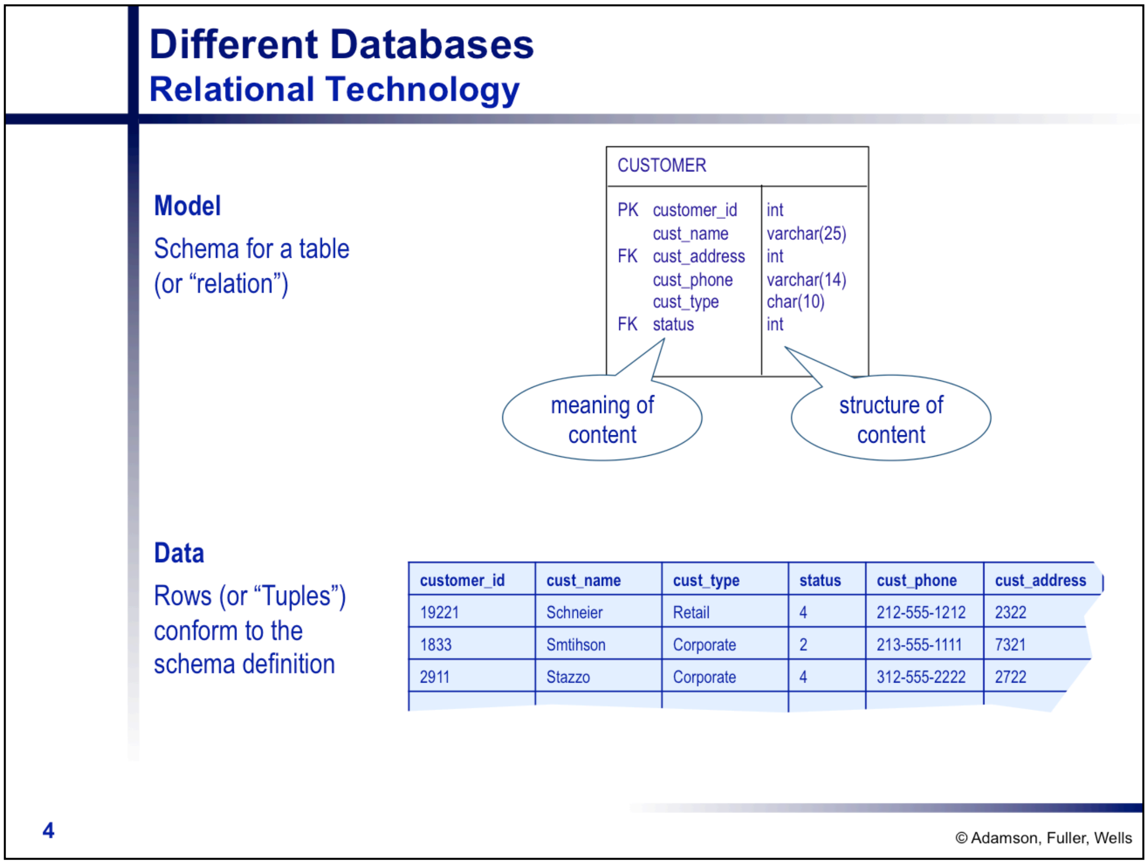
Dave Wells (dwells@infocentric.org)

Course Contents

Part 1

Big Data is Different

NoSQL Technologies
Big Data Challenges



The relational model imposes structure on content.

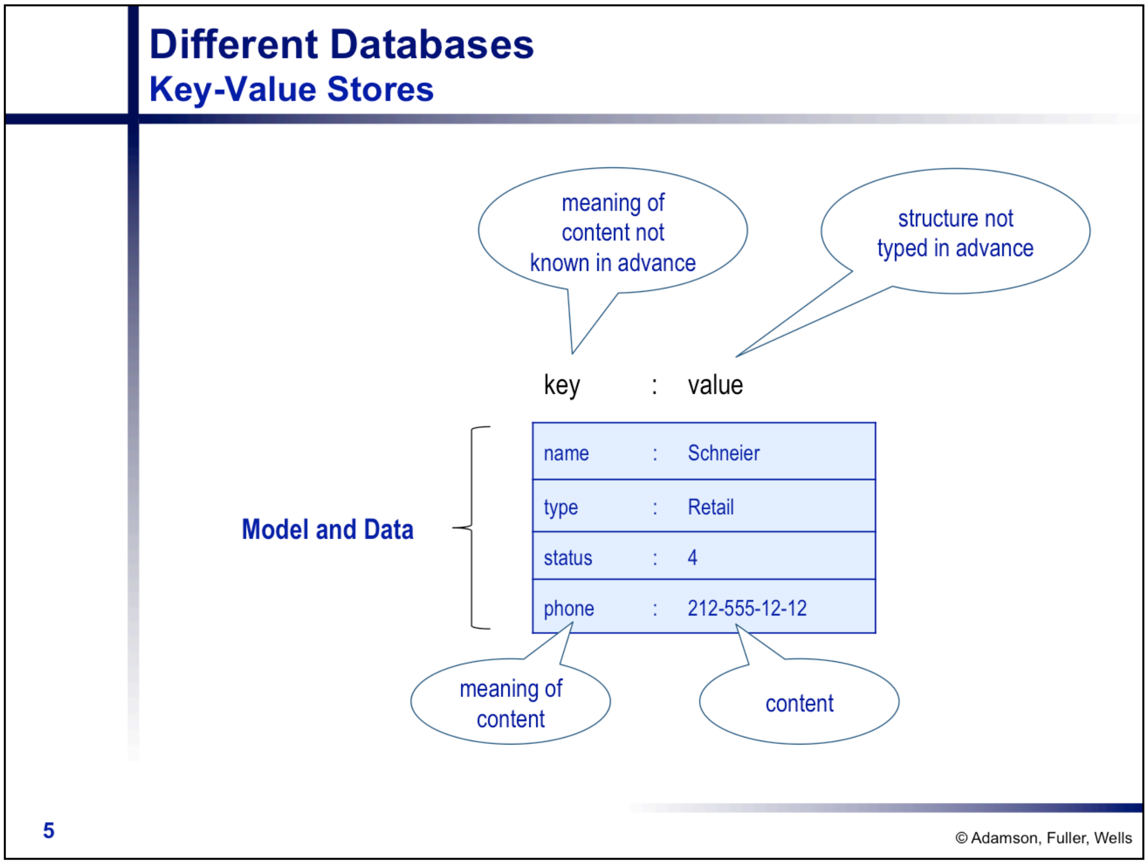
Specifically, a relational design specifies two important characteristics of the data contained in each table.

1. It describes the **physical structure** of the data to be stored. This includes columns, data types, whether values may be left out, etc.
2. It describes the **business meaning** of the data to be stored. A column name describes what kind of data it contains in business terms. (e.g. Customer names, customer addresses, etc.)

This paradigm has some important implications.

- Structure and meaning are determined in advance, before rows can be recorded. That is, a table must be defined before a row can be stored.
- The structure of a row is rigid; each row must adhere to the declared structure of the table.

While this predictability may be advantageous from an application development point of view, it can also be seen as unreasonably rigid.



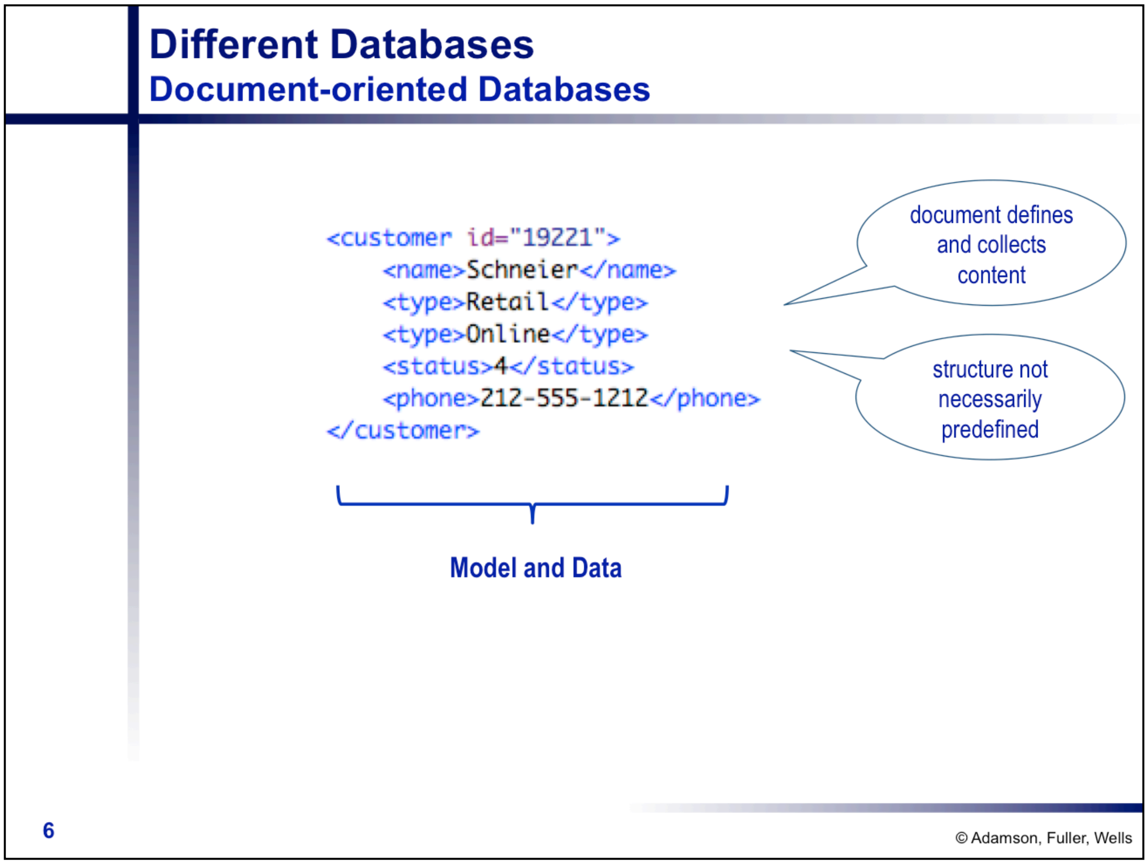
There are several forms of non-relational storage. The most fundamental is the key-value store, which stores key-value pairs (also sometimes called attribute-value pairs.).

Key value pairs are exactly what the name implies– combinations of keys and values. The example above shows several key value pairs that describe a customer.

Notice that the structure (data types) and content (business meaning) of pairs are not defined in advance. **Anything** can fit in a key value pair.

This is extraordinarily flexible. On the other hand, if you want to find something, you have to know where to look. The model is in the data itself.

Key value stores will be explored in Part 3 of this course.



Another form of non-relational storage is the **document-oriented database**, or **document database**.

These data stores track self-contained documents.

Documents do not have pre-determined structure. Instead, they have internal, self-defined structure.

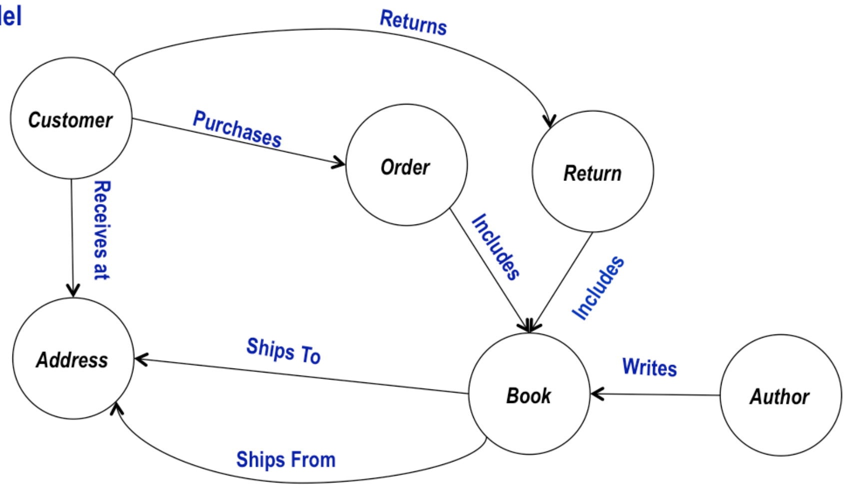
Documents that describe a single business concept, like a customer, are referred to as a collection. The documents in a collection are not required to have the same structure.

Documents can also contain repeating attributes (called arrays) or even other documents. They may also refer to one another, but it is up to applications to be sure these associations are accurate.

We will explore document-oriented data stores in Part 4.

Different Databases
Graph Databases

Model



In a graph database, the objective is to explore how things are related to one another. This may sound similar to relational technology, but graph databases do this in a fundamentally different way.

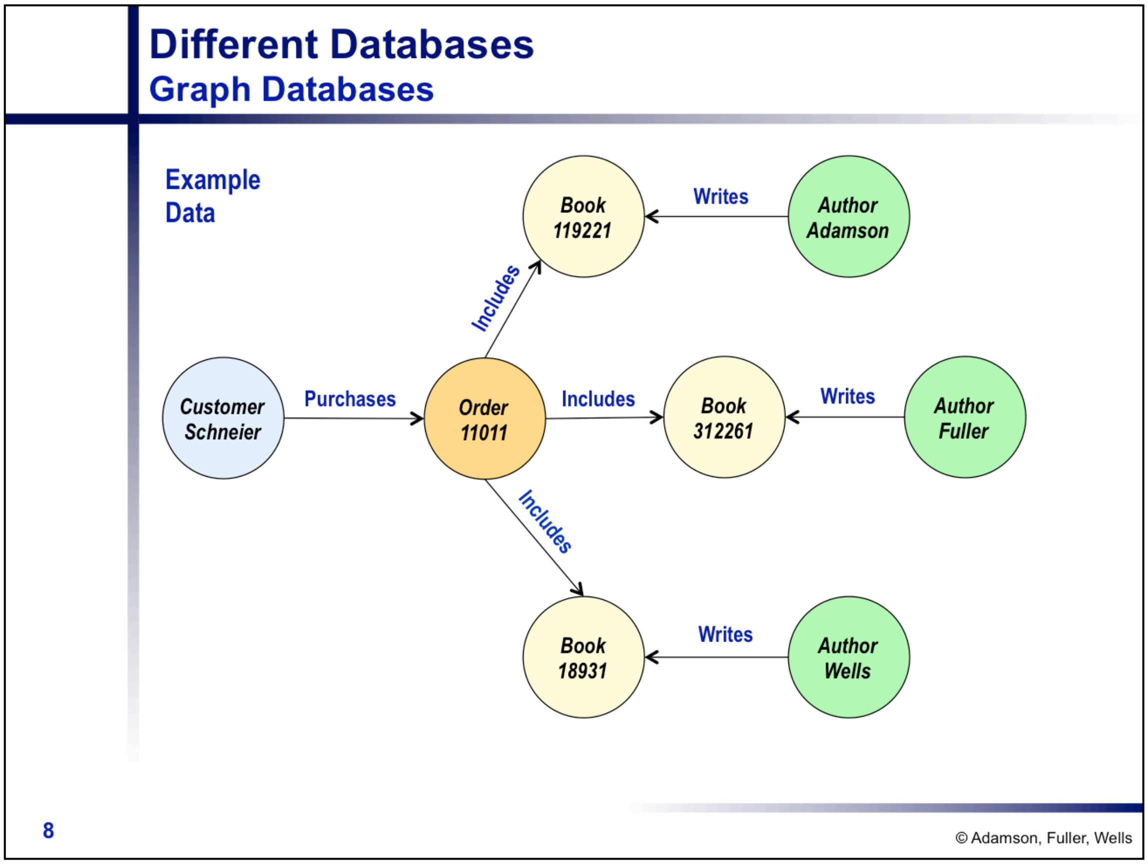
In a relational database, the focus is on things. Relationships are enabled through primary key/foreign key relationships and an expensive join process.

In a graph database, there are nodes and properties, which are similar to entities and attributes. However, there is third concept called an “edge”. Edges describe the associations between nodes.

In a graph database, elements are stored with links to any associated elements. This is called “index free adjacency” and it allows the exploration of relationships without indexes.

The illustration above depicts the relationships and nodes that describe a customer order.

.



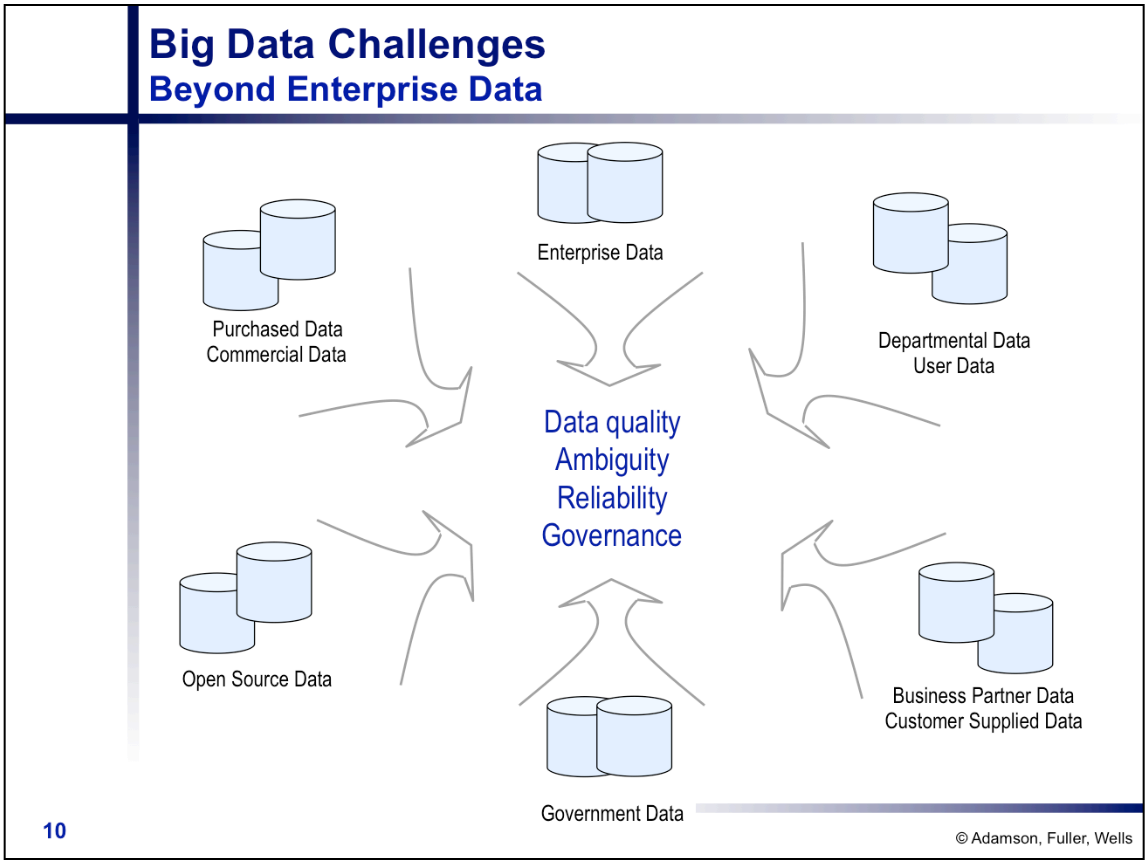
This illustration shows some example data that might be captured based on the model from the previous page.

Different Databases
Summary of Database Technologies

Technology	Terms	Characteristics
Relational	Table Column Key	Heavily typed Uniform records
Key Value	Array Key Value	Loosely typed Flexible
Document	Collection Document Field	Self defining Nested structures
Graph	Node Edge Property	Relationship focus

9

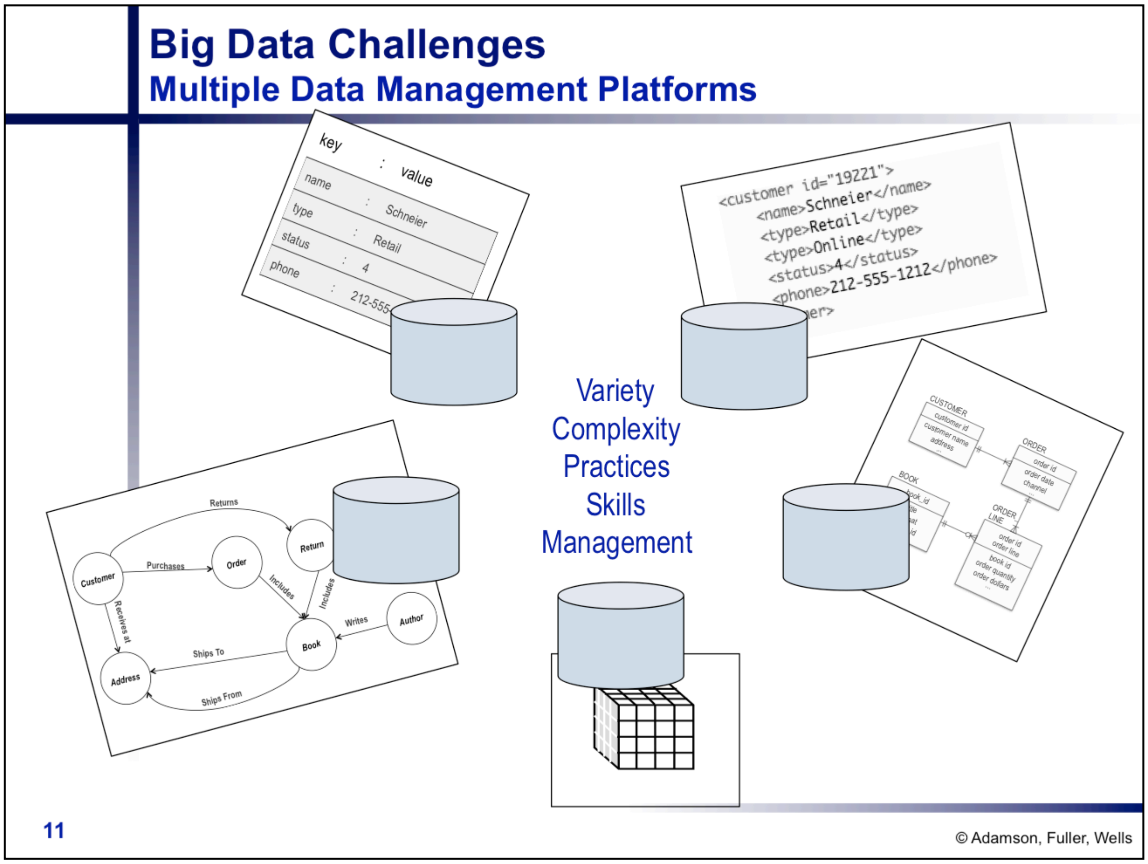
Each type of data store has its own unique vocabulary. Some of the fundamental terms are summarized in the table above.



As businesses move into the world of big data, we face several new challenges.

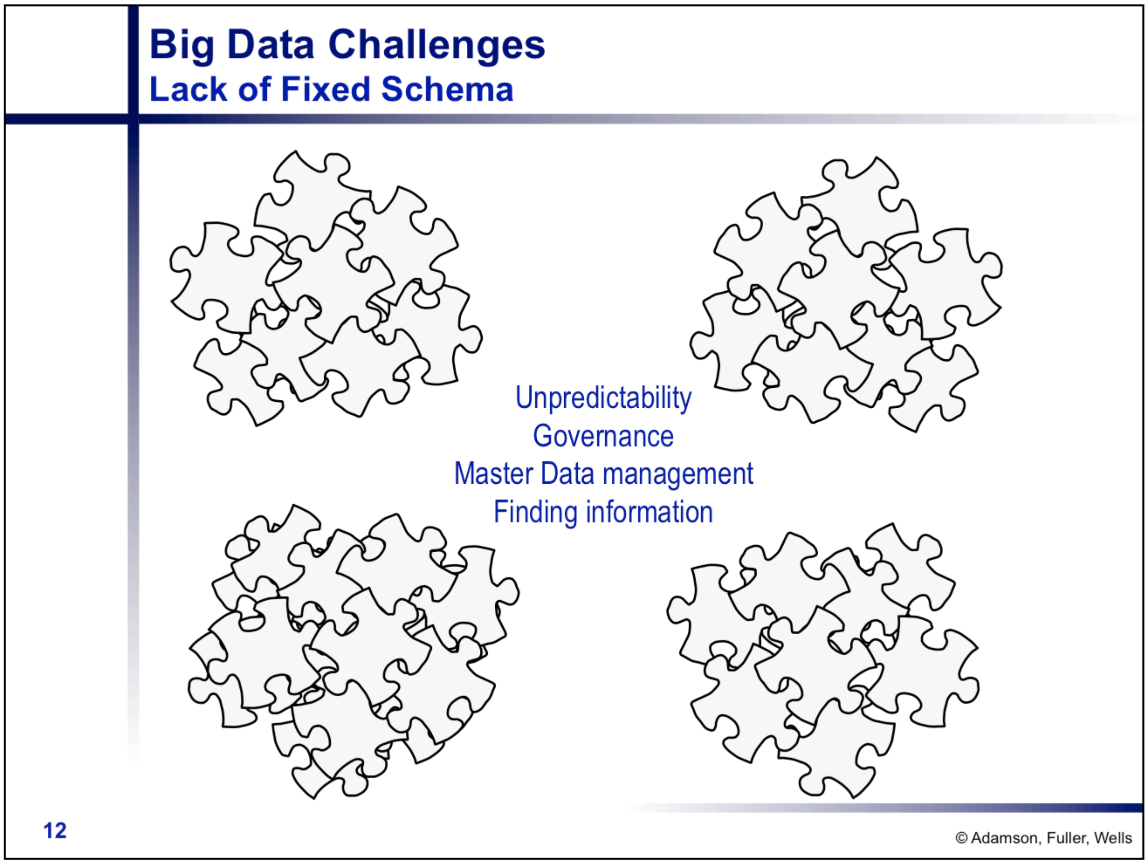
As previously discussed, big data expands the scope of information asset management beyond the scope of enterprise data. As we move to include data from other sources, traditional data management challenges become magnified.

- How do we understand, manage, and cope with quality issues for data that is not created by enterprise systems?
- How do we work with data that has ambiguous meaning?
- Is it possible to build solutions when external sources cannot provide data elements on a consistent or reliable basis?
- How to we extend data governance programs to cover these new forms of data? Are the rules and standards the same or different?



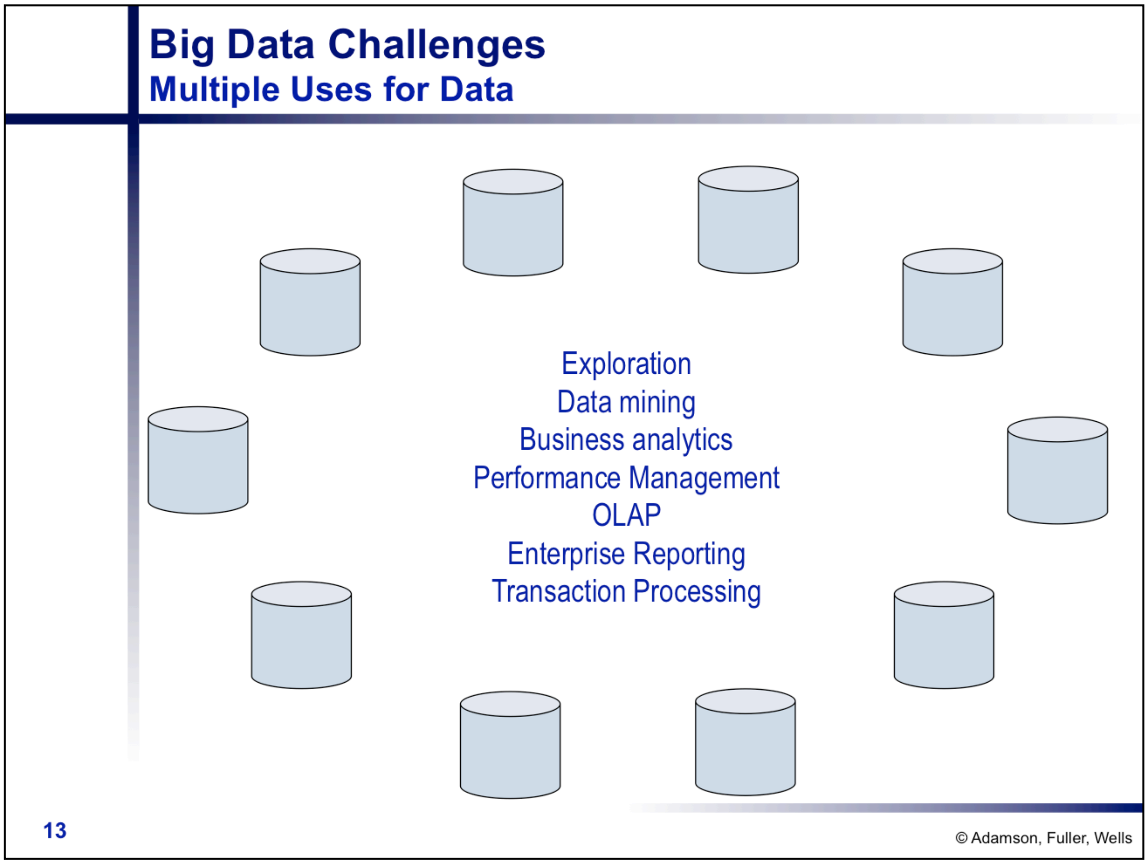
The fact that data can be managed on numerous platforms introduces new challenges as well.

- It is necessary match data's characteristics and consumption to a variety of different platforms, each with unique strengths.
- Information asset management must deal with additional complexity driven by this variety. Simple flows from source to target are no longer the rule; instead there are many kinds of data stores and flows to plan and manage.
- Practices and skills are different on each platform. Organizations must internalize new sets of best practices, and be sure that sufficient expertise is available.
- Data management platforms require different maintenance and management procedures, with must become part of the operational profile of the BI program.



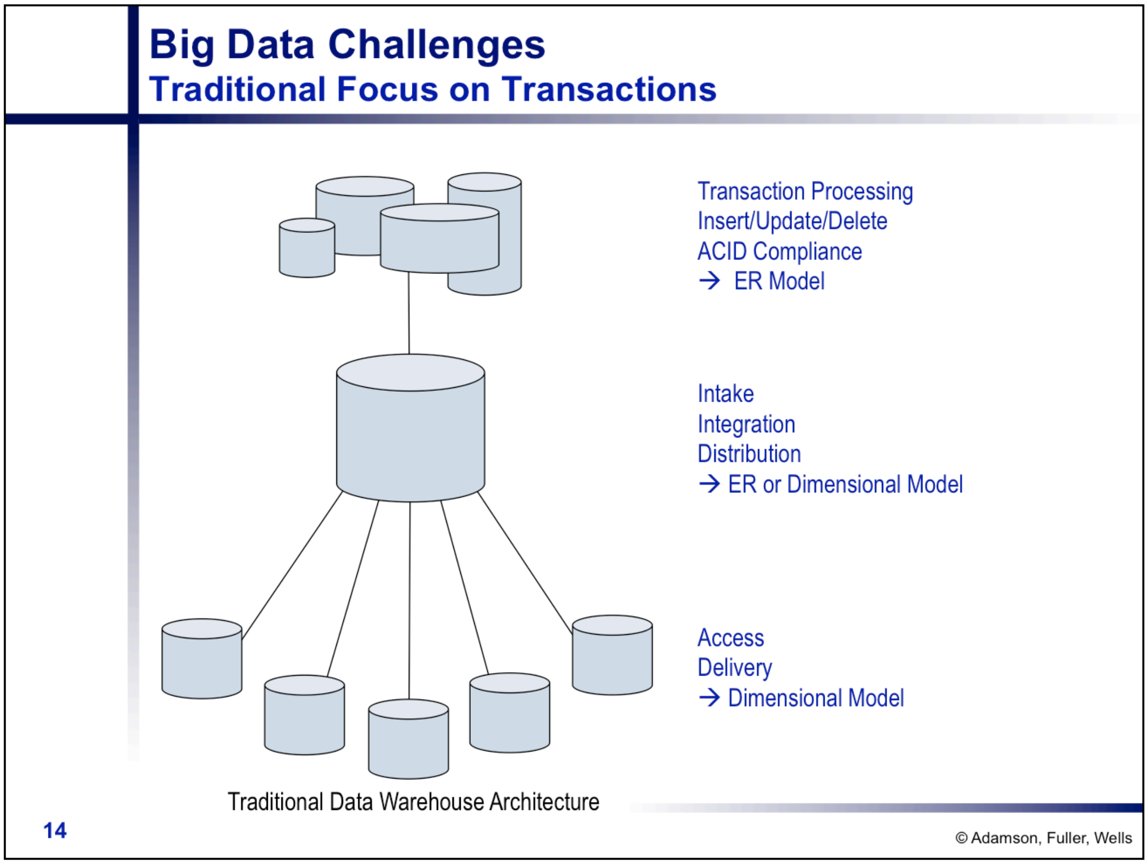
Many of the new forms of data management are not based on the notion of a single pre-defined schema that specifies structure and consent.

- Does this mean there is no data model?
- How does the business cope with data when its format cannot be predicted in advance?
- Is it possible to govern such data, or apply master data management techniques?
- How do you find information with no schema?



With big data analytics, the traditional uses of data have increased.

- Information asset management has grown in scope from simple transaction processing to incorporate a variety of BI and Analytic functions.
- BI programs have an increasing number of responsibilities, and must manage dependencies across functions.

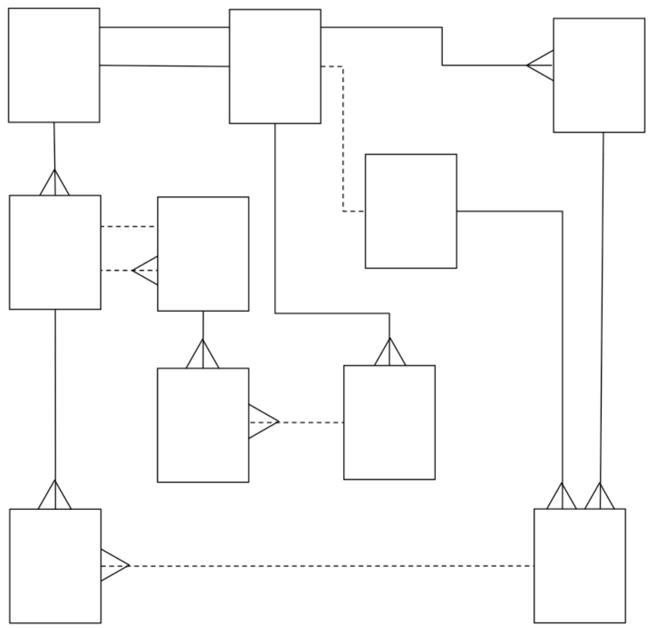


Traditional BI programs focused on enterprise transaction data and leveraged relational and multi-dimensional storage.

The purposes of information management and associated best practices are founded on this aging view.

In Part 2, we will look at new purposes served by data stores in the age of big data.

Big Data Challenges
Relational Perspective



```
select
  cust_id,
  prod_id,
  sum(order_doll
from
   customer
   order_header
   order_line
   customer_acco
   cust_address
   address
where
   customer.cust_
   and order_head
   and order_head
   and customer_
   and address.a
   and customer.
   and order_hea
   and order_lin
group by
   customer.cust
   product.prod
order by
```

15

Most of us begin with a relational perspective on data.

In a relational data model:

- Schema is known in advance
- Schema tells us the shape of data (e.g. data types, optionality)
- Schema tells us the meaning of data (e.g. "Customer", "Author" etc.)

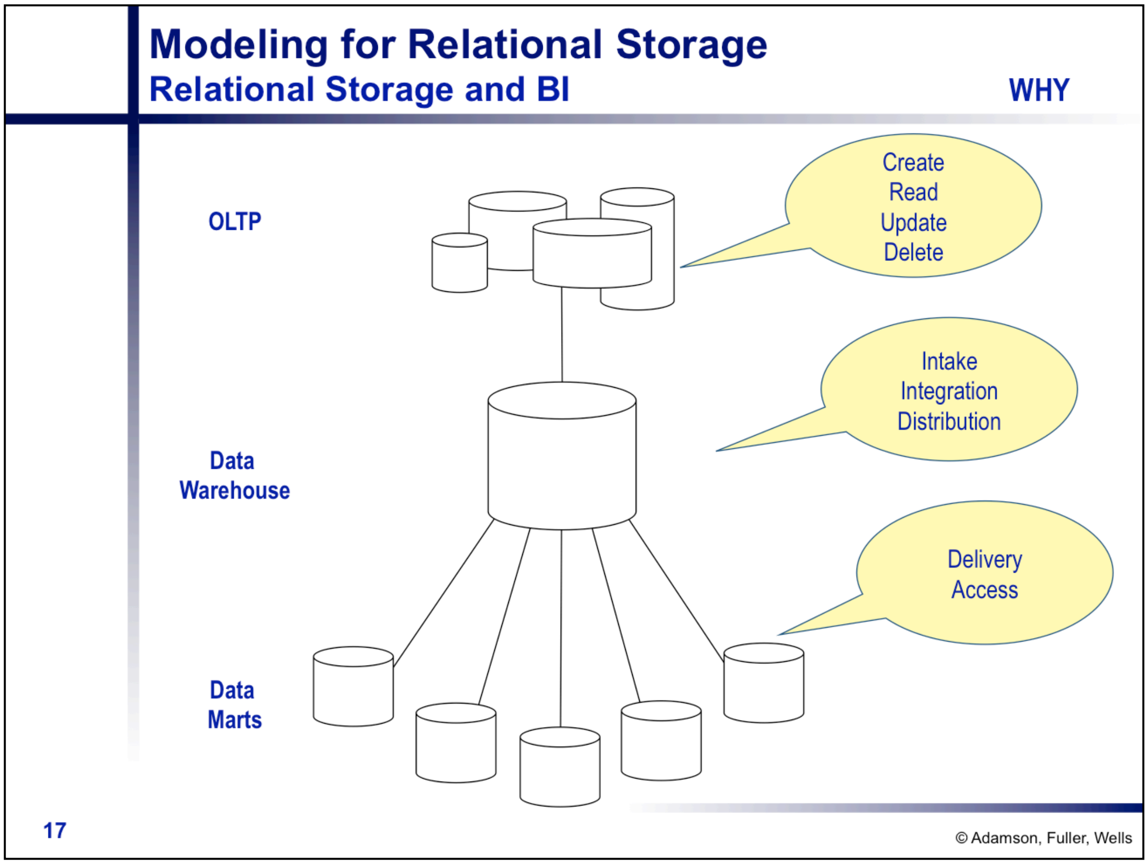
In the world of big data, these assumptions are not guaranteed.

This course will help bridge this gap. Starting with the familiar, we will look at the kinds of things data describes. Then we will explore how these things may be represented in alternative technologies.

Part 2

Modeling and Data

Models
Modeling for Relational Storage
Modeling for non-Relational Storage
Complimentary Approaches



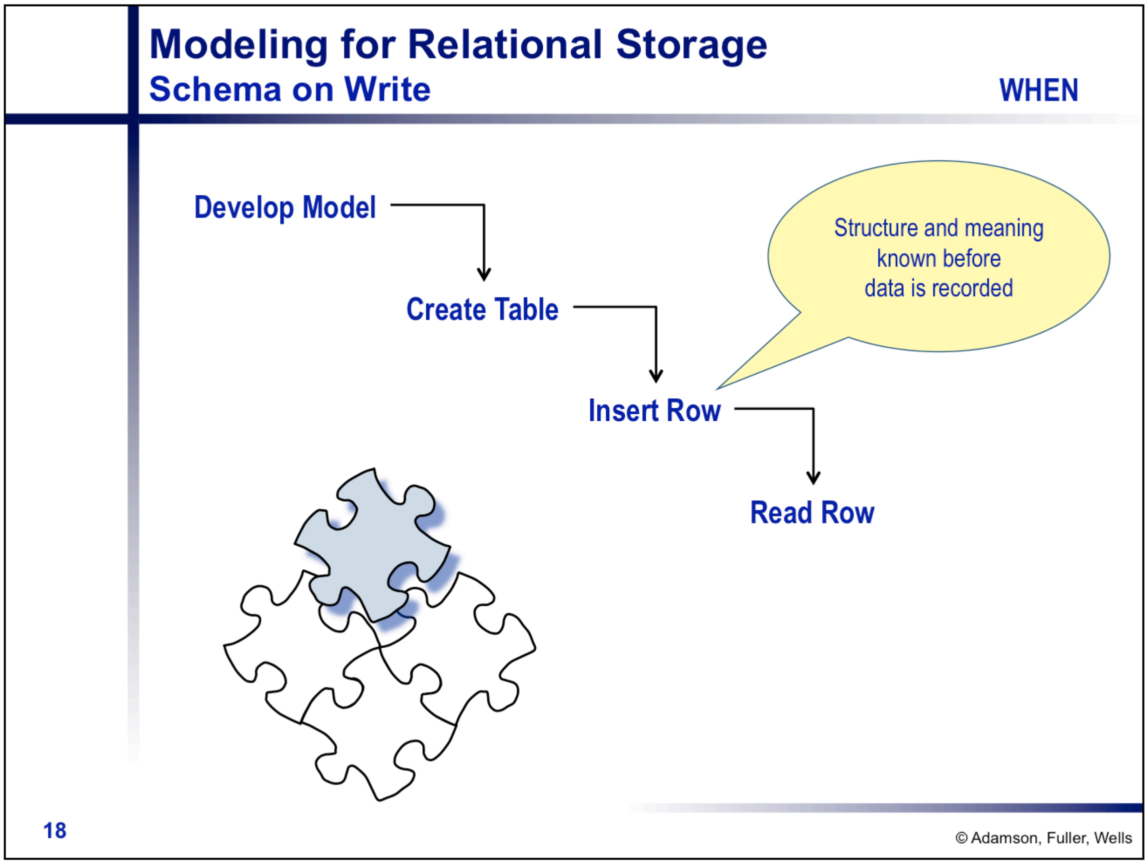
Before we look at the ins-and-outs of non-relational storage, let’s step back and look at relational storage.

Traditionally, relational storage has been used for transaction processing (OLTP) and for data warehousing.

Within the world of transaction processing, it is necessary to support the various **inserts/updates/deletes/reads** that take place, while guaranteeing referential integrity. Relational storage and associated ER modeling techniques evolved to meet these needs.

Data warehousing brings with it a new set of needs. Data stores are used to facilitate **intake** and **integration** of OLTP data. Here, this is shown in a hub-and-spoke model. The hub must also **distribute** data to several data marts. The data marts themselves **deliver** information to business people, and support self-service **access**.

Within the world of data warehousing, both ER and dimensional modeling techniques are employed to serve the purposes of intake, integration, distribution, delivery and access.



As you saw in Part 1, relational storage calls for a model to be defined prior to recording data. In other words, *the table bust be defined before you can record a row*.

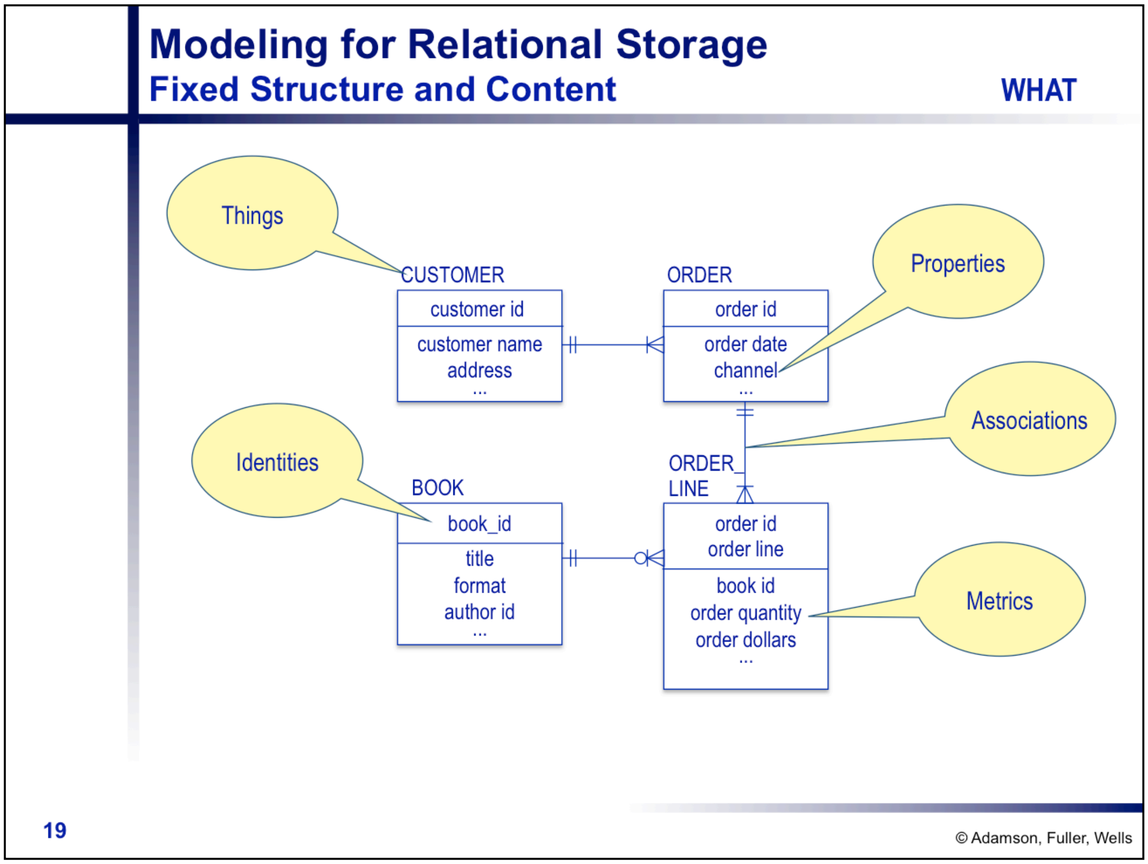
This paradigm is often referred to as **schema on write**. The word "schema" refers to the form of the data – its structure and content. "Schema on write" means that the structure and format of the data must be declared prior to actually recording any data.

The advantage of schema on write is the predictability:

- When it is time to record data, we know exactly where it goes
- When it is time to access data, we know exactly where to find it

The disadvantage of schema on write is its lack of flexibility:

- We have to know what is important about data in order to record it
- There is no flexibility in what we record. Every row of every table must have the same structure.



When modeling for relational storage, there are five fundamental kinds of information we attempt to capture:

Things In a logical ER model, things are represented as entities. In a third normal form (3NF) physical model, the are represented by tables. In a dimensional model, we call them dimensions.

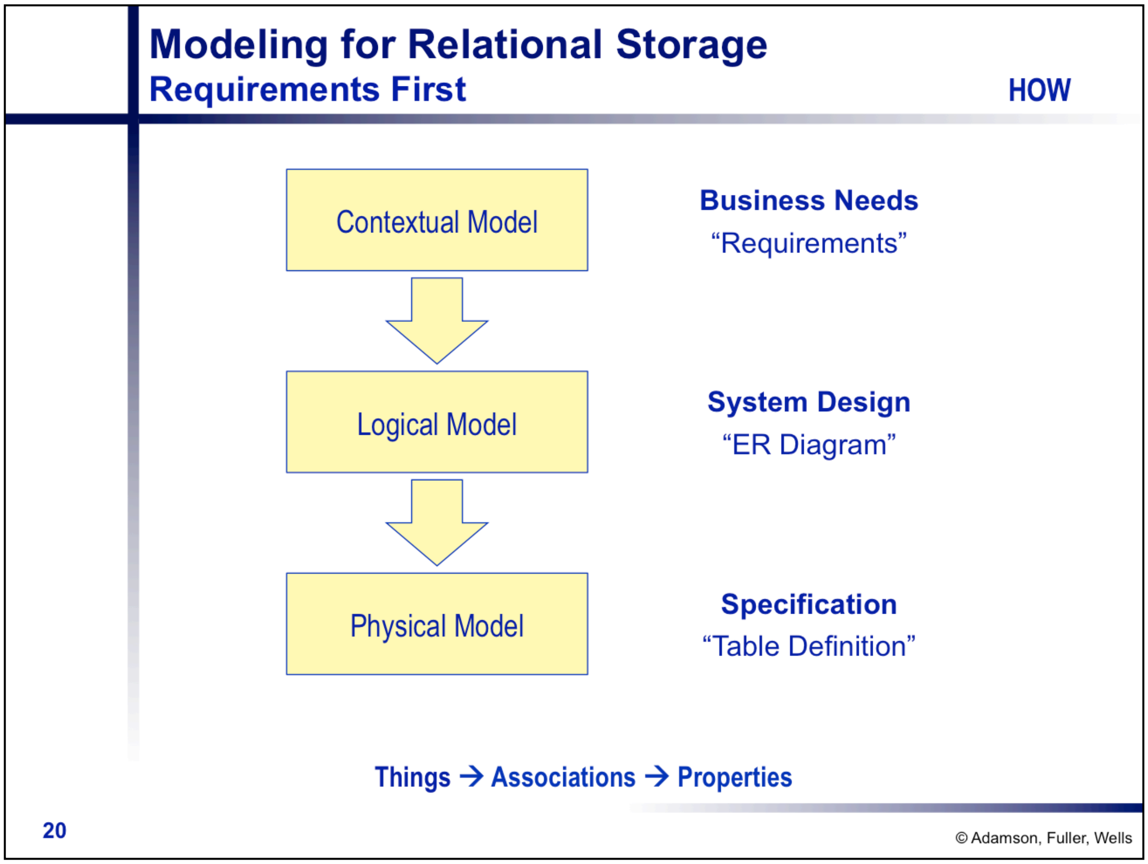
Properties In a logical ER model, properties are represented by attributes. In a 3NF physical model, they are columns. In a dimensional model, we call them dimension attributes.

Identities are attributes that uniquely identify things. In an ER model, we call them UID's. In a physical model, they are called primary keys or alternate keys. In a dimensional model, we call them surrogate keys or natural keys.

Associations link things together in a business context. In an ER model, we call them relationships. In a physical model, we call them primary-key/foreign-key relationships.

Metrics quantify an activity or business process. In an ER model, they are a subset of attributes. In a dimensional model, they are called facts.

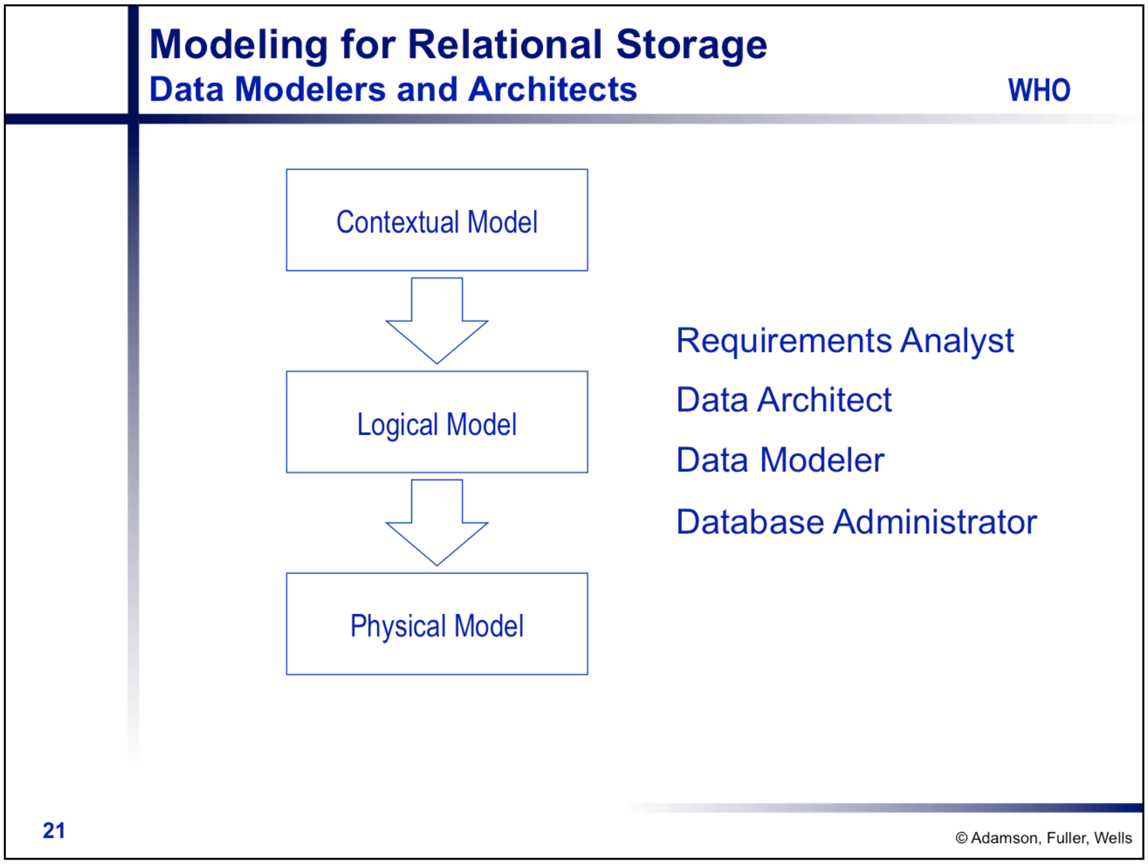
In Parts 3-5, we will show how these fundamental concepts are handled in non-relational data stores.



Modeling for fixed, relational storage begins with an understanding of business requirements before proceeding to a spec for a solution. Typically, this is done in three stages.

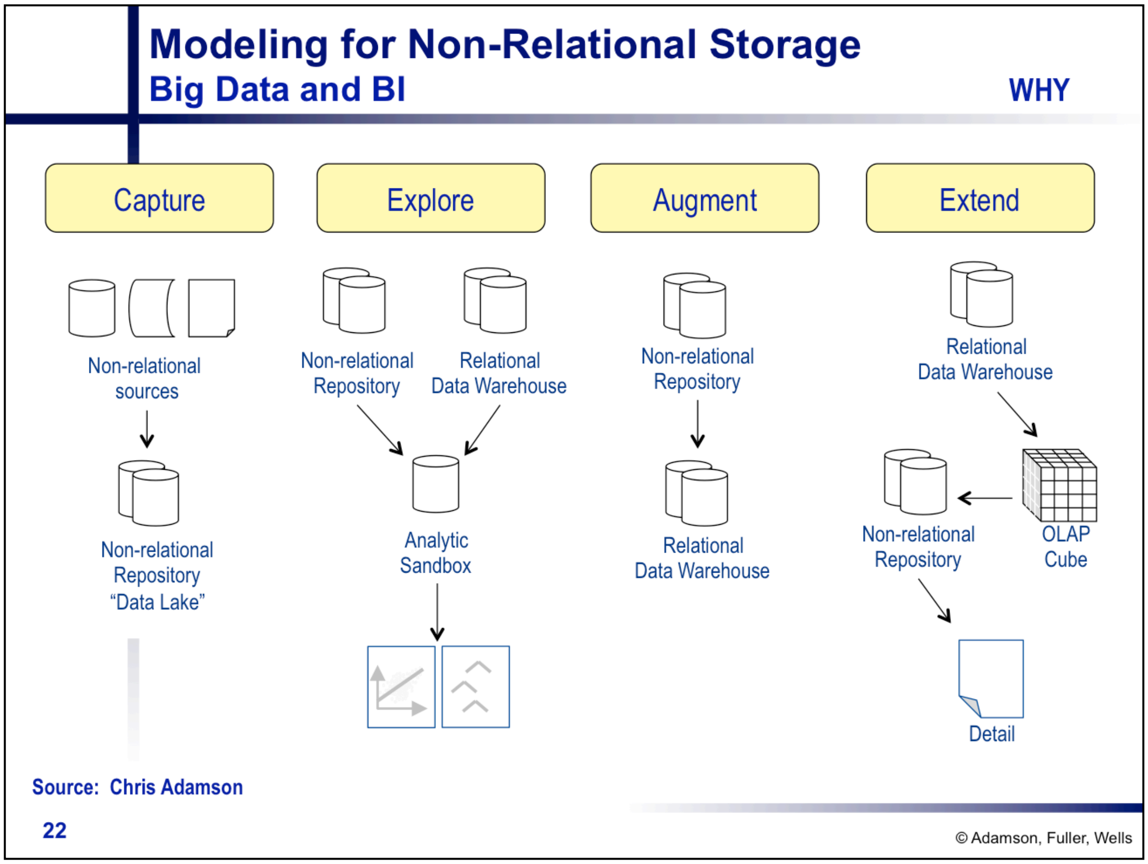
- The first step is creation of a **conceptual model** – a high level representation of needs. One common example of a conceptual model is a "subject model" – a high level representation of 10-12 major business subjects and their relationships.
- The next step is a **logical model**, which specifies functional characteristics of a a solution. An ER model is a common form of logical model.
- The last step is to develop a **physical model**. A physical model is the specification for physical storage. It contains definitions of tables, columns, data types, and so forth.

Another way to look at this progression considers the characteristics of the model. At the conceptual level, the primary focus is on things. At the logical level, there is an increased focus on associations and properties. At the physical level, all characteristics are defined. There is a progression from Things → Associations → Properties



For projects that use relational storage, division of labor a byproduct of the top-down approach. Typical roles at each level are:

Conceptual	Requirements Analyst
	Data Architect
Logical	Data Architect
	Data Modeler
Physical	Data Modeler
	Database Administrator



The traditional BI categories of "intake, integration, distribution, delivery, access" are not fit to describe use cases for modeling in the world of big data.

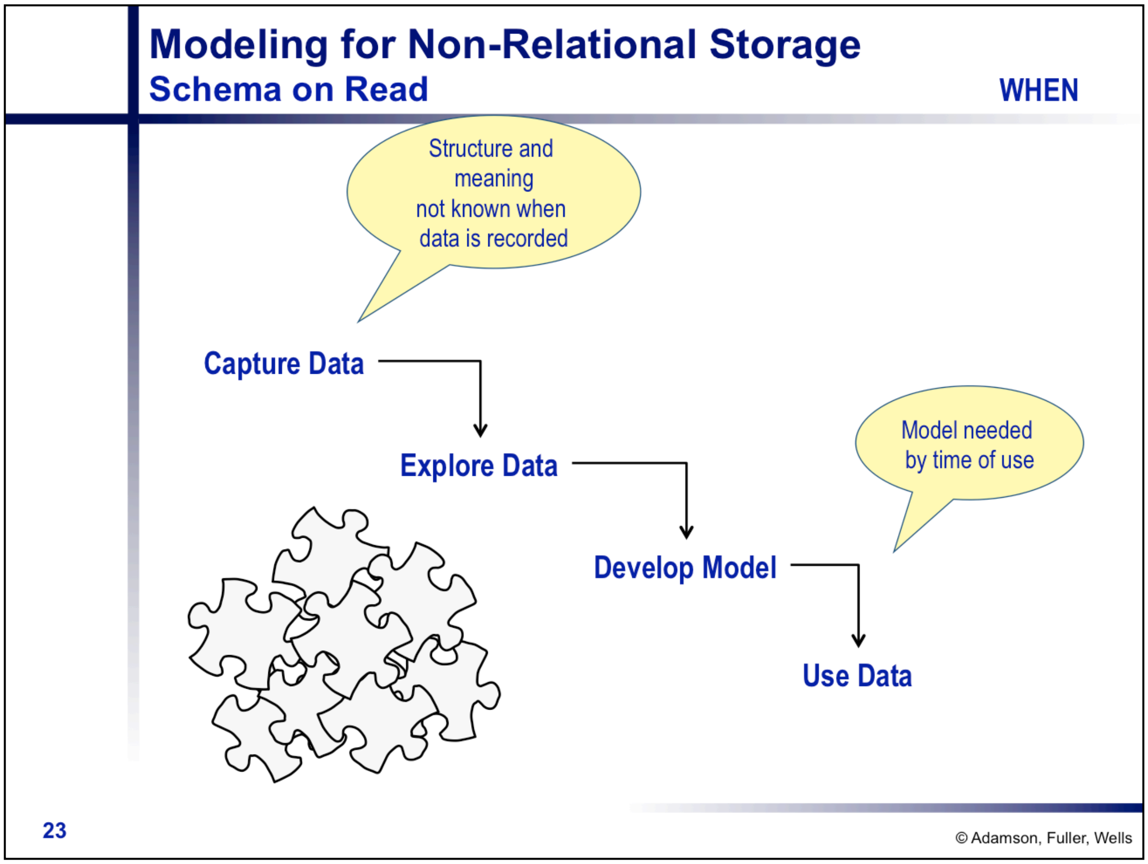
Chris Adamson identifies four uses for non-relational data stores:

Capture Provide a repository that can be used to bring new information sources under the control of an information asset management. The relational model may not be fit to the task if the structure of data is not known, varies, or does not lend itself to relational storage.

Explore Create a data store that can be used to explore the data to find business value. This may be an analytic sandbox used to develop a predictive model, or a repository used to link new data to enterprise data to search for useful information.

Augment Use non-relational storage as a staging area to bring new data elements into the data warehouse. This can only be done once exploration has identified value.

Extend Maintain data in a non relational extension of the data warehouse. Users who drill to detail in the data warehouse can then drill through to non-relational detail, such as an XML record of a transaction.



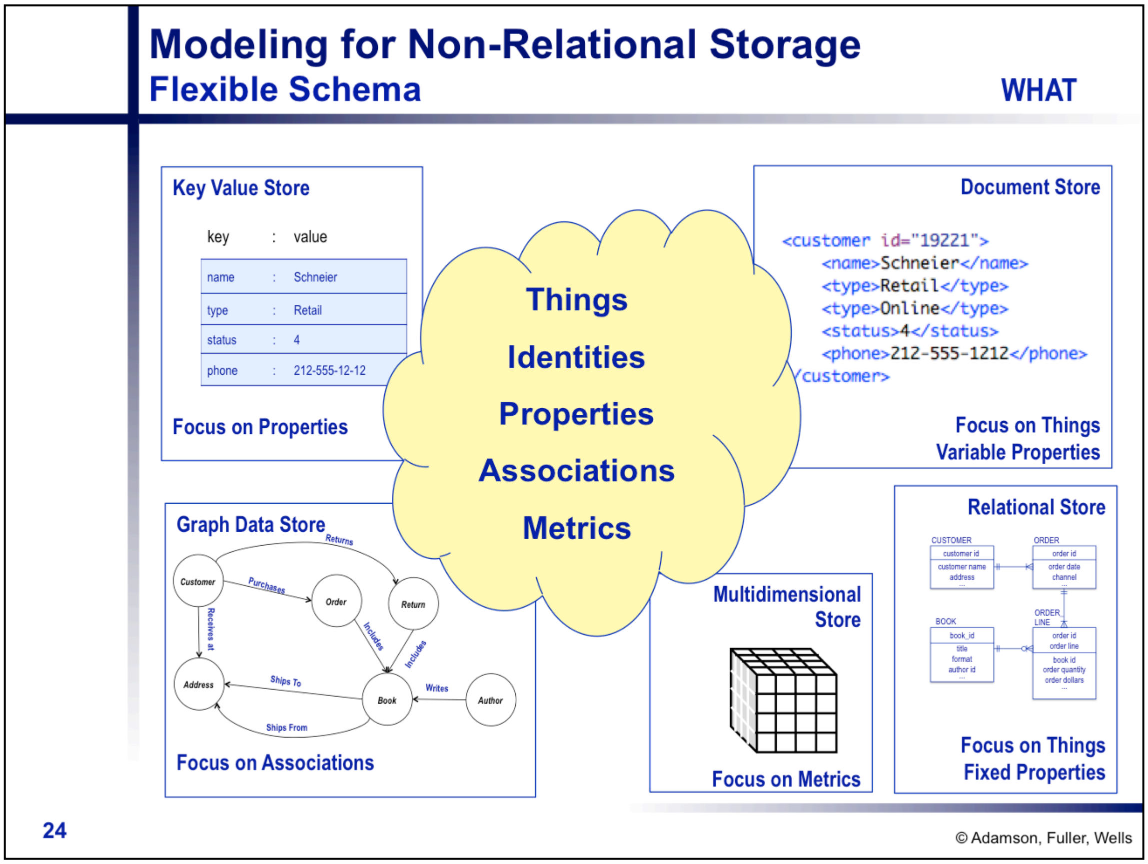
When it comes to non-relational data, schema-on-write is not always practical. Perhaps we don't control the data that is flowing in. Even if we do, we may not know exactly where the value lies. A predefined structure (or schema) does not fit the bill.

With non-relational storage, it is possible to capture data and then learn about its structure (or schema) after the fact.

Of course, when we query the data for business purposes, we have to be able to tell the DBMS what we want and where to get it. At the time of reading the data, the schema must be known. Hence the term **schema on read**. We will explore schema on read in the context of Map Reduce in Part 3, Key-Value Stores.

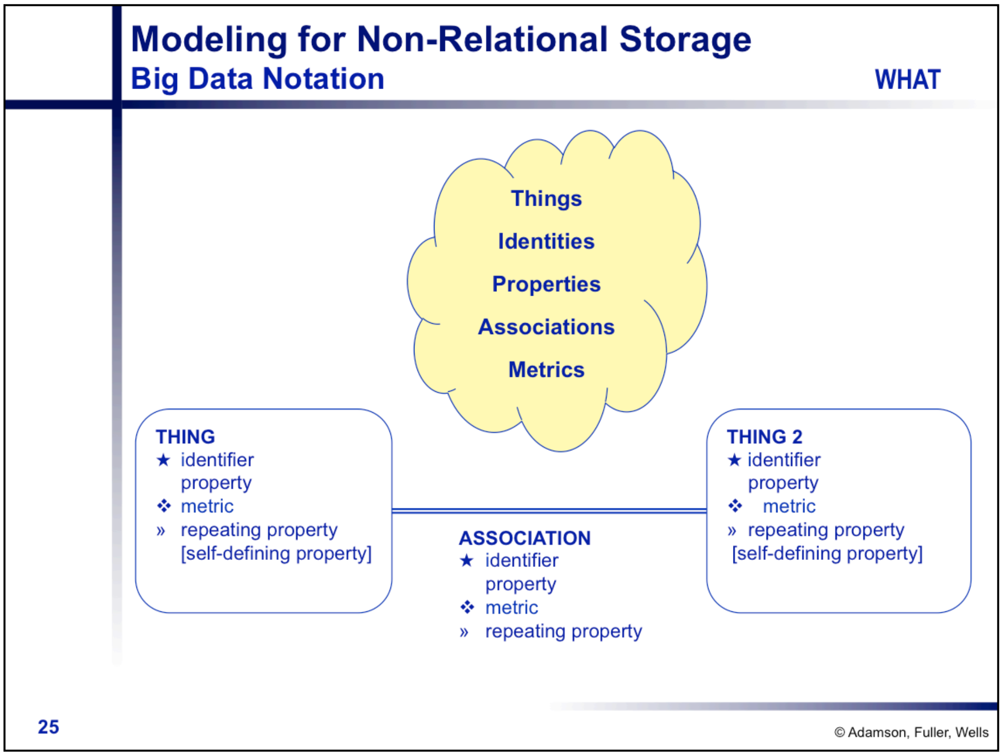
Schema on read places a heavy burden on applications that use data. They must tell the DBMS what they are looking for and where to find it. For example, if you wish to see all products of a certain color in a key-value store, you need program a request to look for keys called "Color" with value "Blue", as well as keys called "Blue" with value "True". There is no "column" called color that you can query.

Note: It is also possible to employ a schema-on-write paradigm with non-relational data, though the DBMS may not enforce it. For example, a document repository can accept customer orders regardless of format (schema on read) or enforce a single format programmatically (schema on write). However, the document store will not impose this format; it must be managed by the application that records the data. More on document



When we develop a model to describe non-relational data, we capture the same kinds of elements previously described.

Each storage paradigm emphasizes different characteristics of data.



This course will use a standard notation to construct logical models for information stored in relational and non-relational databases. This notation is used to capture the business value of the information store.

Things Represented by rounded rectangles. The name of the thing appears in all capital letters.

Identities Indicated by a star: ★
They can appear in things or beneath associations.

Properties Listed in lower case and may contain spaces. They can appear in things or beneath associations.

Data types are not specified for properties.

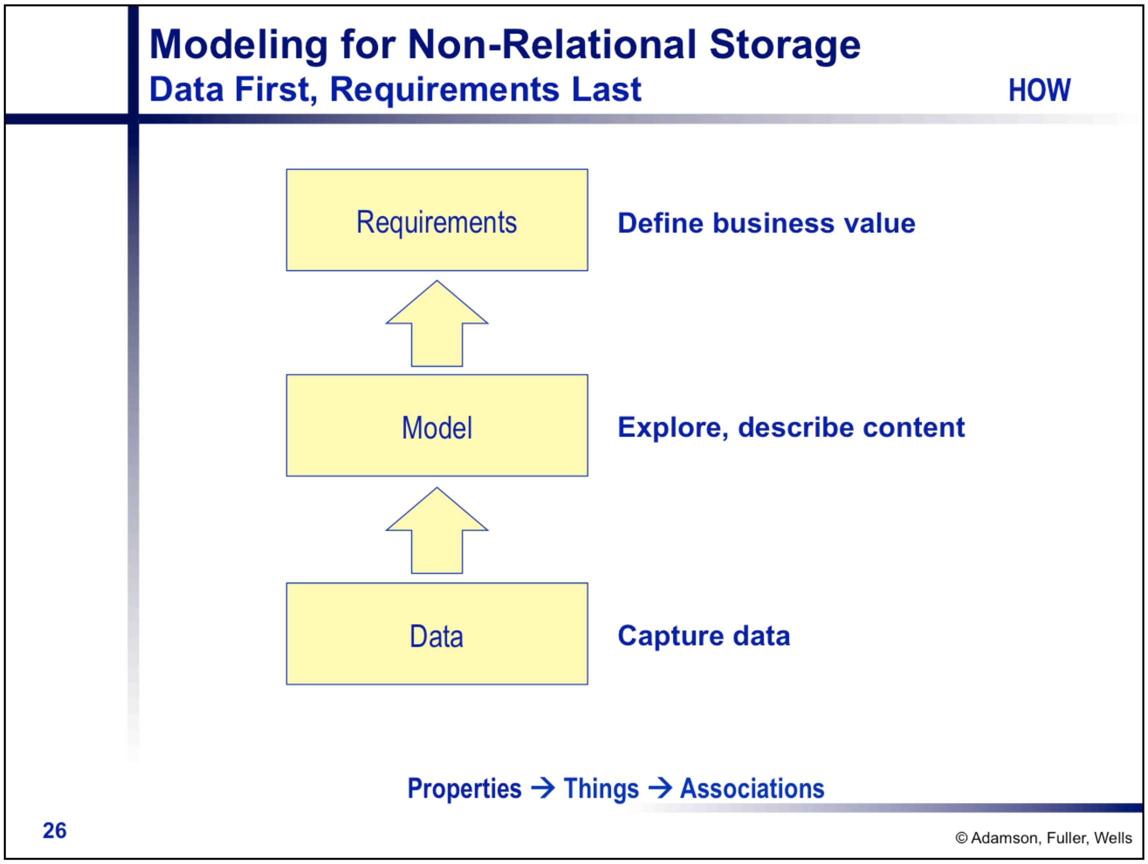
Properties can repeat in some non-relational stores. Repeating properties are listed with a double pointer: »

Properties may be self-defining. These are represented using square braces with a generic name such as **[property]**

Metrics Indicated with four diamonds: ❖

Associations Indicated by double lines between things: ═══

Associations may have identifiers and properties.



How do we produce a model of non-relational data? The process varies, but often we begin with raw data. A log file dumped into a key value-store, for example, represents the unknown. What is there? Is it valuable to the business?

We explore this data to understand it, and in the process develop a model. For example, a business analyst and programmer write programs to explore the log, and find that it contains timestamps, geographic coordinates, and identifiers. A model begins to emerge.

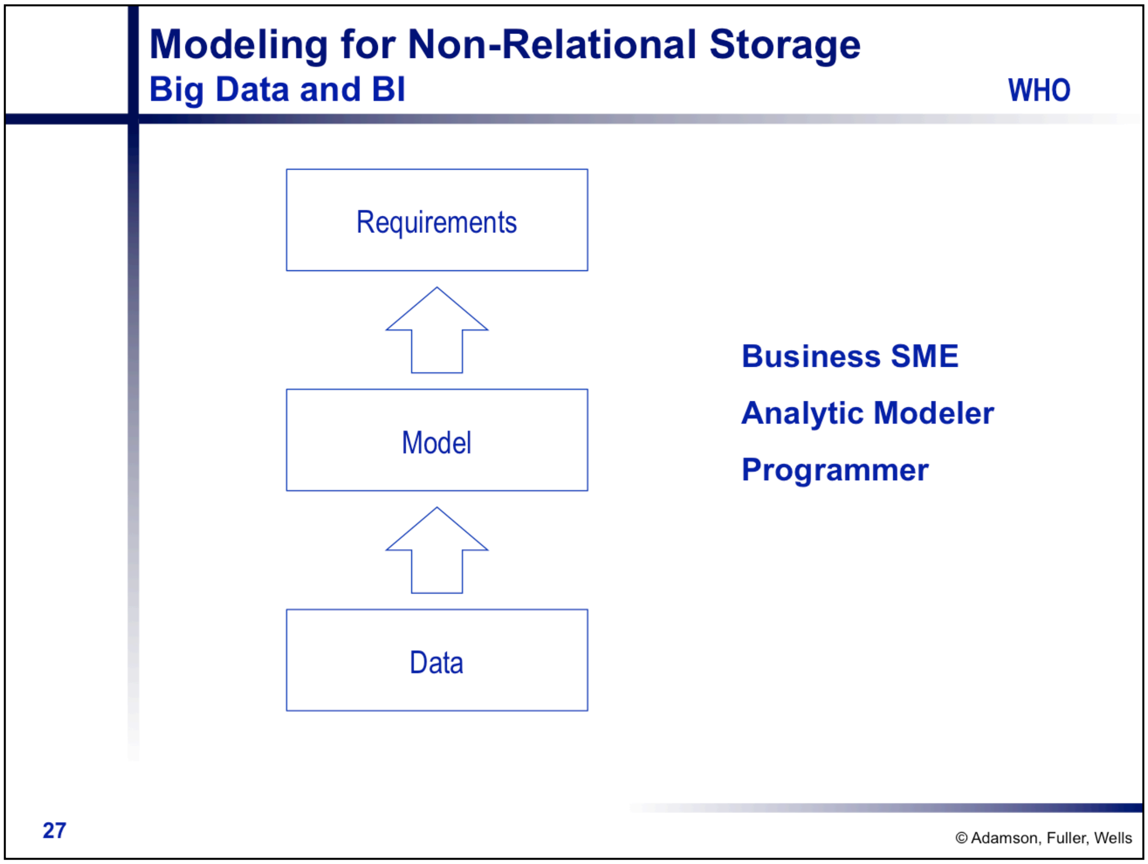
How is this information useful to the business going forward? The last step is to define the business value. For example, it may be useful to append this information to a package delivery record in the data warehouse.

Another way to think about how this process differs from the relational process discussed previously is to look at how the model comes into focus. In the relational world, we observed a progression like this:

Things → Associations → Properties

In this non-relational example, we started with properties and eventually organized them to describe things and associations.

Properties → Things → Associations



Development of models for non-relational data is a fundamentally different process from relational model development. So it should come as no surprise that it requires different roles as well.

Business Subject Matter Expert The business subject matter expert is a key participant in modeling for non-relational storage. These people may not write the code necessary to explore the database or draw the models. But they are the only ones that can truly know when we have found something useful in the data.

Analytic Modeler Someone who knows how to apply statistical analysis and data mining techniques to construct models that describe or predict useful business concepts or events. These people are often called data scientists. They know how to work with data, but need business experts to point them in the right direction.

Programmer Many non-relational data stores are accessed programmatically. For example, a map-reduce operation in Hadoop is usually written in Java or Python.

Complimentary Approaches
Relational and Non-Relational Data

	Relational	Non-Relational
WHY	Intake Integration Distribution Delivery Access	Capture Explore Augment Extend
WHAT	Rigid structure	Flexible structure and content
WHEN	Schema on write	Schema on read Schema on write
HOW	Top Down	Top Down Bottom Up
WHO	Modelers Architects	Programmers Data Scientists Business SME's

28

This page summarizes the key differences we have identified between relational and non-relational storage.

Non-relational technology does not replace relational technology, it compliments it. It is easy to lose sight of this amidst all the excitement of new technology.

In 2013, TDWI attendees packed a ballroom to hear Ken Rudin talk about how Facebook was using Hadoop to generate business impact.

During the presentation, Rudin shocked the audience by pointing out that one of his top priorities was to implement a relational database. Hadoop was great for some things, he said. But when it came to analyzing facts and dimensions, relational is the technology you need.

Part 3

Key-Value Stores

Key-Value Stores Defined
Key-Value Data Representation
Use Cases
Examples

Key-Value Stores Defined
The Basics

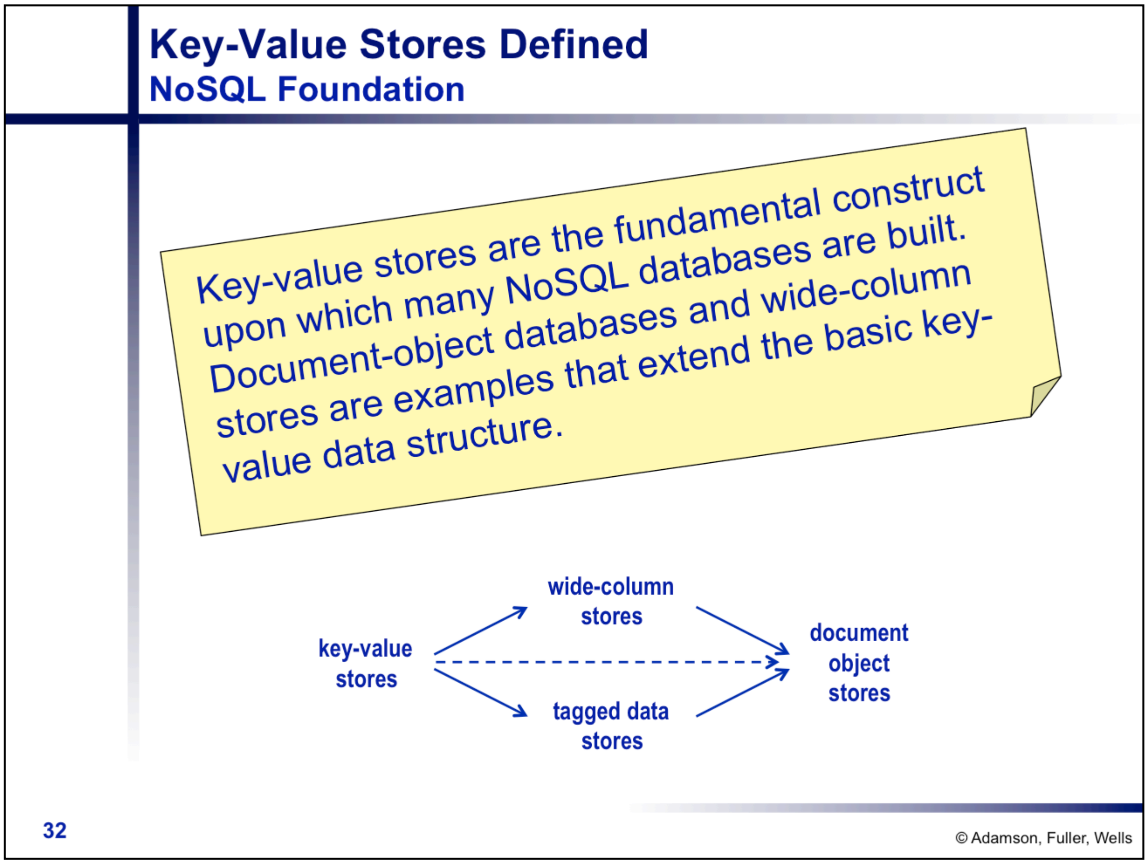
> "Key-value stores are probably the simplest form of database management systems. They can only store pairs of keys and values, as well as retrieve values when a key is known."
>
> db-engines-com

- Associative Array – Data is represented as a collection of key value pairs
- Each key-value appears only once in the data collection
- Extensions support sorting of keys for range queries and ordered processing

31

Key-value stores are the simplest form of database management systems. They can only store pairs of keys and values, as well as retrieve values when a key is known.

These simple systems are normally not adequate for complex applications. On the other hand, it is exactly this simplicity, that makes such systems attractive in certain circumstances. For example resource-efficient key-value stores are often applied in embedded systems or as high performance in-process databases.



The real power of key-value stores in big data is adaptability. Because the concept is simple it can easily be adapted to represent nearly any kind of data. The KVP concept has become the basis for many NoSQL databases. Big table data stores and big table clones (wide-column stores) are use KVP as a foundation, as do document databases.

Key-Value Data Representation
Representing Things

Key	Value
Location1	Los Angeles
Location2	Chicago
Location3	Dallas
Location4	Atlanta
Location5	Raleigh-Durham

Key	Value
FirstName	Donald
LastName	Duck
Species	Anthropomorphic White Duck
Occupation	Actor
Genre	Cartoons

Key	Value
Blue	2
Green	3
Red	2
Yellow	2

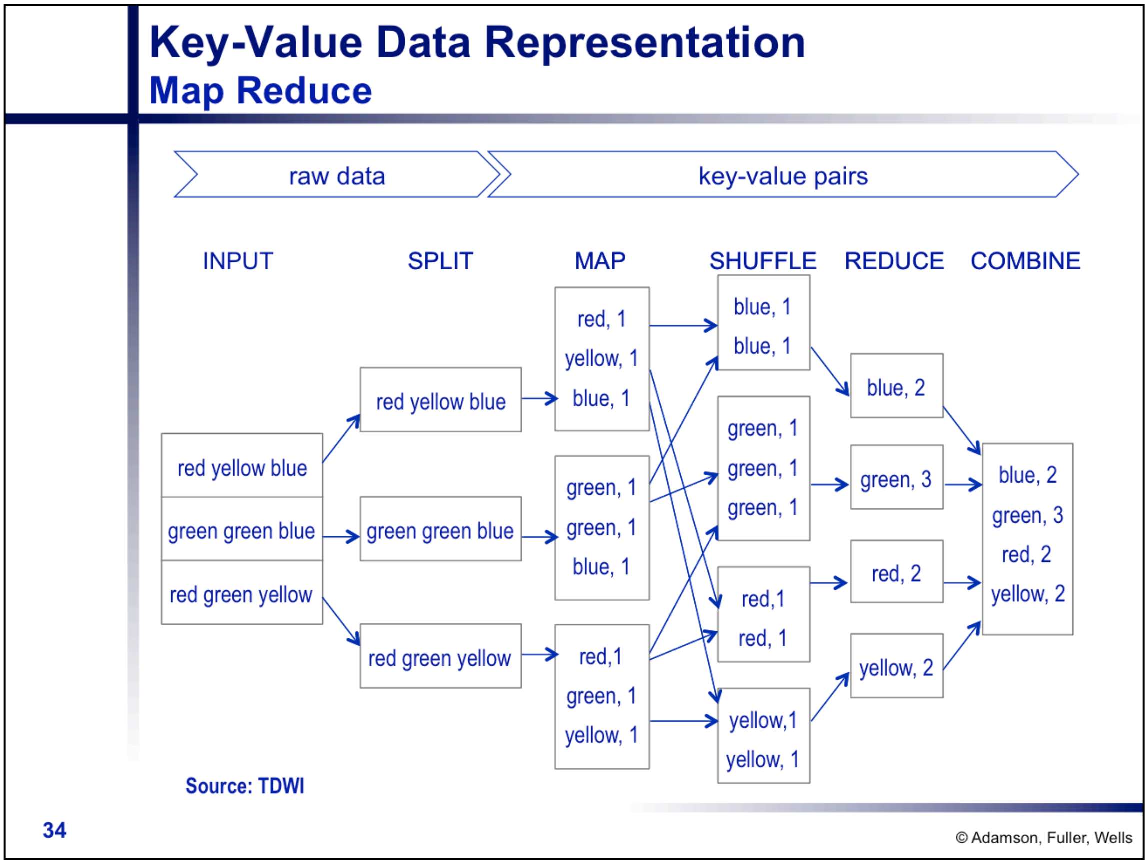
33

The examples shown here illustrate three of the many possible applications of KVP data structures.

The example on the left has data for several occurrences of location. Each location is a thing with a key – location 1, location 2, etc. – and a value that names a city associated with the location. This example shows one thing, many occurrences of that thing, and a single descriptive property – the city name – for each occurrence. The keys, in this instance, identify distinct occurrence of location.

The example on the right has data for one occurrence of a thing – a character – and describes several properties of that thing. The keys in this example name the properties with which data values are associated.

The example on the bottom has data for several categories -- colors. Each category has an associated metric – a count that describes the category. The keys in this example are the categories with which the metrics are associated.



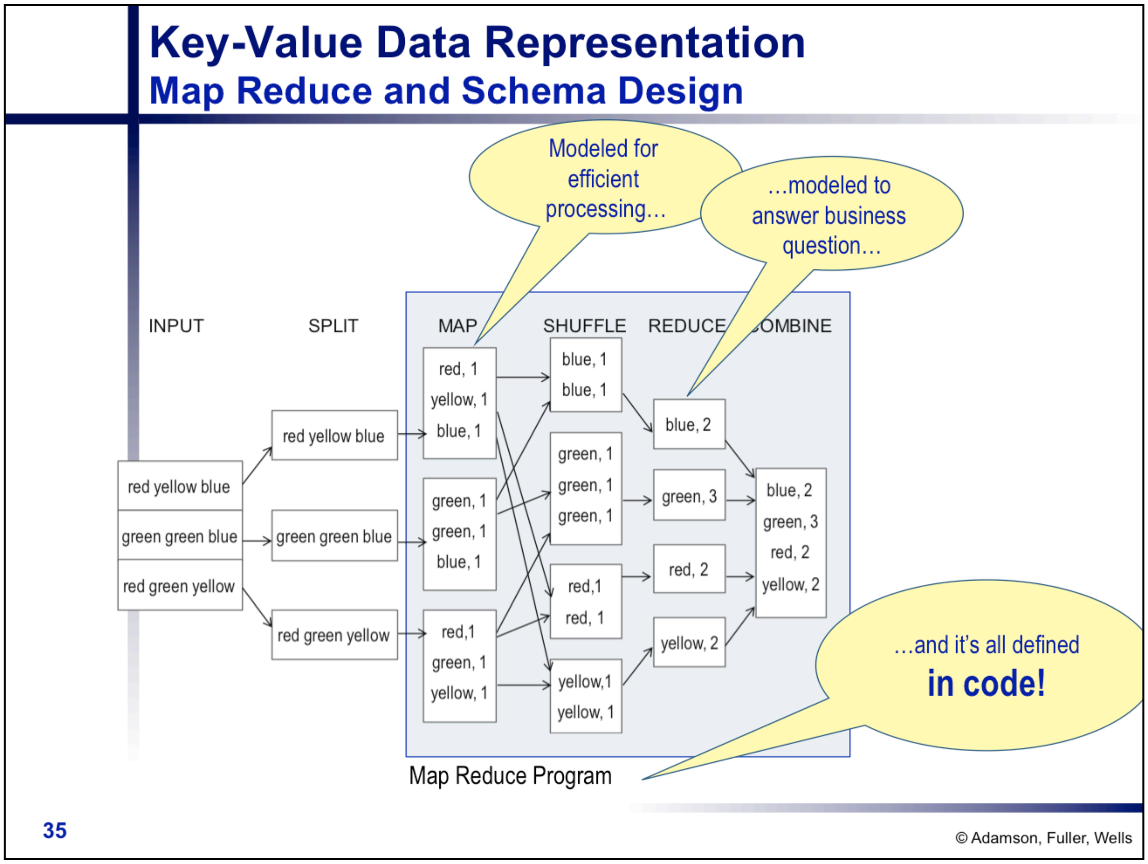
MapReduce is a software framework to process large amounts of unstructured data in parallel. It was originally developed at Google. An open source implementation of MapReduce is part of Apache Hadoop.

MapReduce processing is programmatic, and may involve several phases. Its name comes from two phases that are commonly included. The **map** step separates and structures the data, "mapping" it into key-value pairs. The **reduce** step summarizes to yield an output dataset that is physically smaller than the input.

There may be other operations (such as a shuffle), and there is not always a reduce step.

The example in the illustration comes from TDWI's "TDWI Big Data Fundamentals" course. An **input** file is **split** across multiple nodes in a cluster of computers. Then a MapReduce program processes the data:

- The **map** step imposes some structure on the data, transforming it into key-value pairs that will be useful in processing the data (color and count)
- A **shuffle** redistributes the pairs based on the key (color) in order to optimize the next step
- The **reduce** summarizes the data for each key (color)
- The results are then **combined** to answer the business question



In Apache Hadoop 2.0, MapReduce jobs are defined programmatically, using languages such as Python or Java to make calls to the MapReduce library.

In the example, notice that key-value models are created twice. The mapper reorganizes data around keys that represent colors. This allows the shuffle step to redistribute the data to nodes by color, making the reduce step possible. The reducer produces another set of key-value pairs, in this case a list of colors and their quantities. The final result is output to a file.

In other words: the data is modeled first to support processing, and second to produce a desired output.

Most importantly, all of these key value pairs are **defined in the program** that was written to count colors. None of it is applied in advance; none is imposed by a DBA on the "back end."

MapReduce programs must define the data structures to be processed, as well as the algorithms to process the data data. In the relational world, applications do not have these responsibilities.

Key-Value Data Representation
Representing Identities

Key	Value
Location1	Los Angeles
Location2	Chicago
Location3	Dallas
Location4	Atlanta
Location5	Raleigh-Durham

★ identifier of locations

36

While key-value pair models are often embedded in code, it is important to understand where they are being used to answer business questions.

Our modeling notation can be used to describe the business value (key-value pair output) of a MapReduce program. This includes concepts like identities, attributes, things, associations and metrics.

Here the location example illustrates how identities are represented in KVP data structures. The key is the identity – each of location1, location2, location3, etc. are identifiers. The notation on the right shows how these identities are represented in a data model.

Key-Value Data Representation
Representing Identities

Key	Key	Value
60127	FirstName	Donald
60127	LastName	Duck
60127	Species	Mallard
60127	Occupation	Actor
60127	Genre	Cartoons
60130	FirstName	Mickey
60130	LastName	Mouse
60130	Species	Rodent

CHARACTER
★ character id number

★ identifiers of characters

37

Using the character example and extending from key-value construct to key-key-value illustrates how identities are represented with many records for one occurrence of a thing. Here we see two characters, Donald Duck and Mickey Mouse, each with several rows of data. Recalling the earlier constraint for key-value stores – each key appears only once in the data collection – this construct appears to be a violation. Key-value processing, however, concatenates the two keys and treats them as a single key. Each combined key appears only once in the data collection.

Key-Value Data Representation
Representing Properties

Key	Value
Location1	Los Angeles
Location2	Chicago
Location3	Dallas
Location4	Atlanta
Location5	Raleigh-Durham

characteristic of locations

38

Here the location example illustrates how properties are represented in KVP data structures. The values are the properties showing the city name for each location. The notation on the right shows how these properties are represented in a data model.

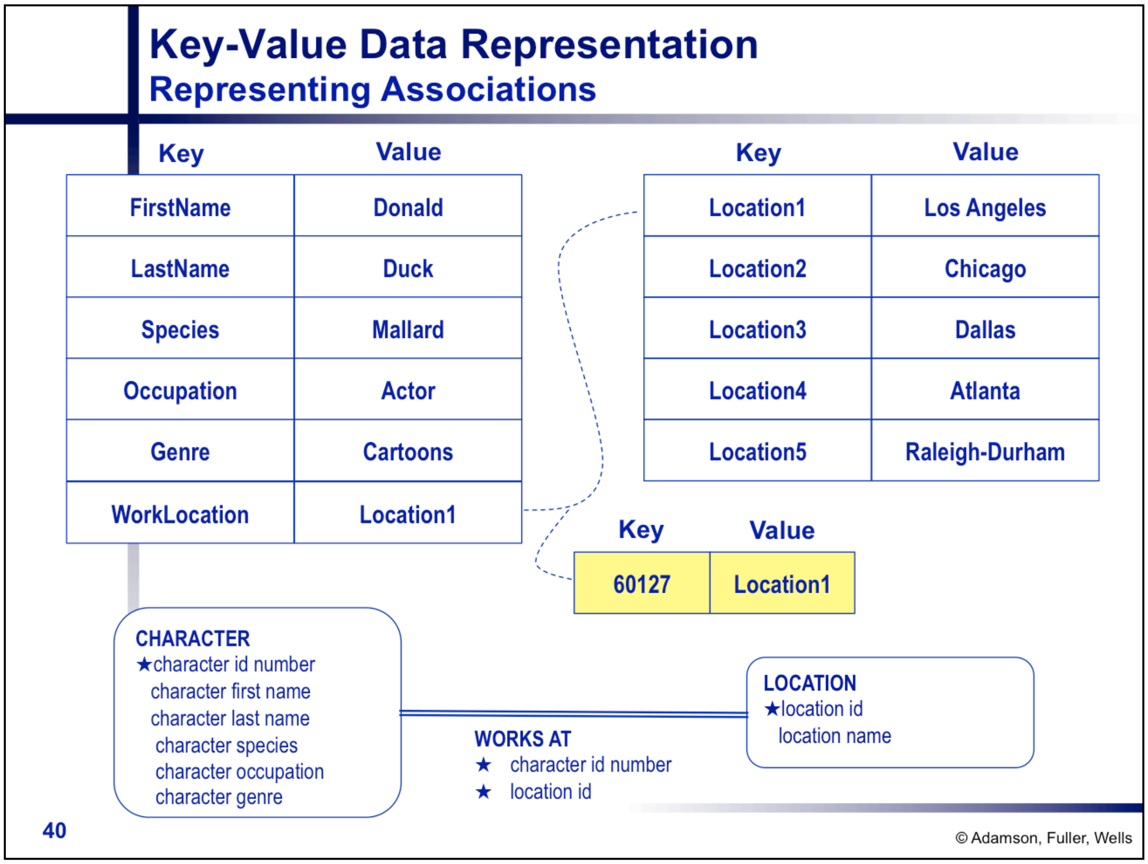
Key-Value Data Representation
Representing Properties

Key	Key	Value
60127	FirstName	Donald
60127	LastName	Duck
60127	Species	Mallard
60127	Occupation	Actor
60127	Genre	Cartoons
60130	FirstName	Mickey
60130	LastName	Mouse
60130	Species	Rodent

CHARACTER
★character id number
character first name
character last name
character species
character occupation
character genre

characteristics of characters

Using the character example and extending from key-value construct to key-key-value illustrates how multiple properties are represented for multiple occurrences of a thing. The numeric key is the identifier of each character, the alphabetic key names the property that is represented, and each value is specific to the unique combination of character and property. The notation on the right shows how this construct is represented in a data model.



Associations are also represented as key-value pairs. In this example, Donald Duck works at Los Angeles – the character with id number 60127 works at location 1. The key-value pair that is highlighted shows how the association can be stored as a key-value pair. Note that *Location1,* a key in the previous examples, has taken on the role of value. The notation at the bottom of the page shows the association in data model form.

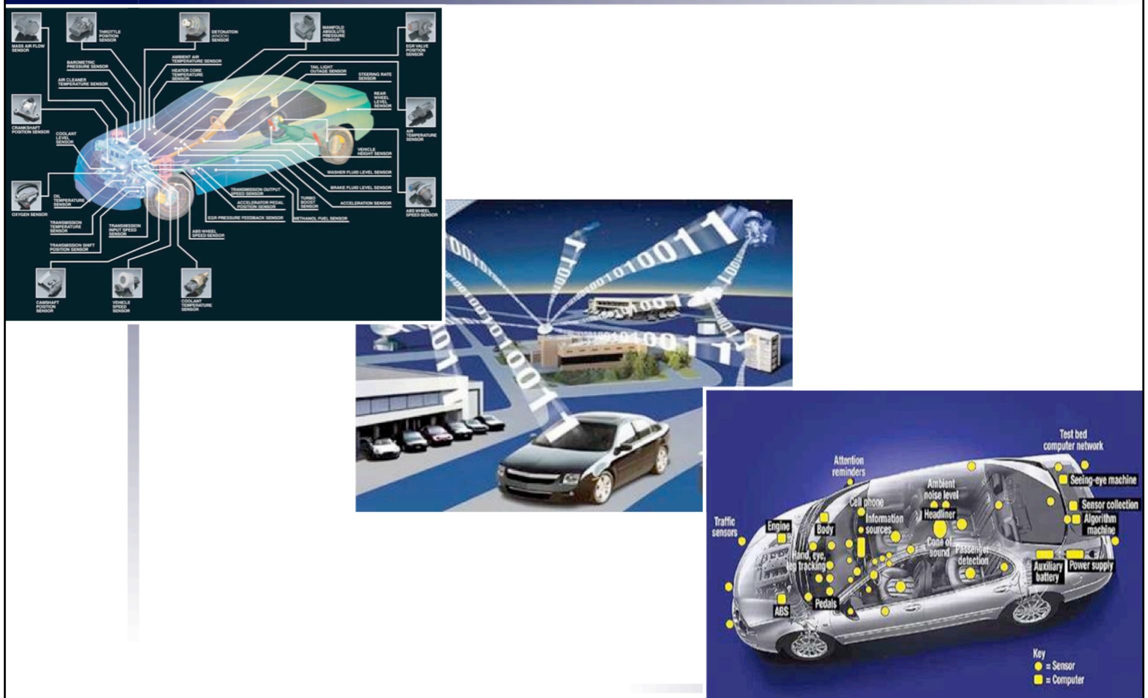
Key-Value Data Representation
Representing Metrics

Key	Value
Rodent	5
Anthropomorphic White Duck	2
Coyote	1
Gorilla	1
Smurf	105
Ninja Turtle	4
Non-Ninja Turtle	4

SPECIES
★species name
species count

When representing metrics (or facts) as KVP data the key is categorical – it identifies a category for which something is quantified. The value stores the quantity. This example counts characters by species. KVP structures frequently collect counts by category, something that MapReduce does particularly well.

Use Cases
Embedded Systems

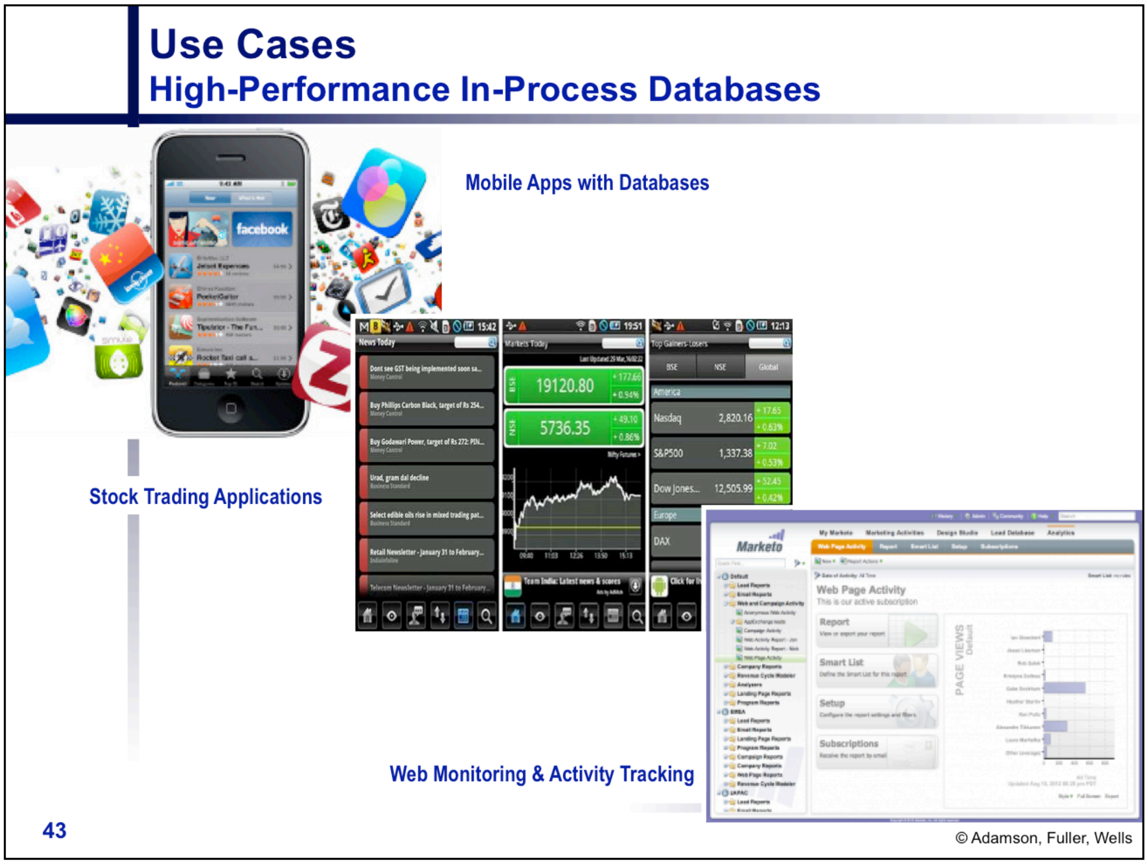


42

© Adamson, Fuller, Wells

An embedded system is a computer system within a larger mechanical or electrical system, typically with real-time processing constraints. Automobiles use many embedded systems such as engine monitoring and cruise control. Industrial embedded systems in manufacturing are often used to reduce emissions and improve energy efficiency by controlling electrical and mechanical devices.

Key-value stores are are commonly used in embedded systems where real-time processing and responsiveness are critical. The simplicity of key-value stores, while limiting for complex applications, makes them a good fit for the demands of high-performing embedded systems.



Similar to embedded systems, in-process databases have high-performance and real-time demands. The simplicity and efficiency of key-value stores makes them a good choice to persist data for applications such as mobile apps, stock trading applications, web monitoring, etc.

Use Cases
NoSQL Foundation

Key Value Stores are the Foundation of NoSQL Databases

Wide Column Stores	Document Stores	Graph Databases
AKA "big table clones" – Store data in records with an ability to hold very large numbers of dynamic columns.	Store collections of documents that have flexible, internally defined structure.	Store and access for graph data structures that map the relationships among things.
Big Table (Google)	MongoDB	Neo4j
Cassandra	CouchDB	Allegro
Hbase	DocumentDB	OrientDB
Hypertable	OrientDB	InfiniteGraph
Accumulo	Terrastore	OntotextGraphDB

Key Value Stores

Hash table -- one key, one value, no duplicates and extreme speedy.
DynamoDB (Amazon), Voldemort (LinkedIn), Azure (Microsoft), Riak, Redis, LevelDB , Dynomite, Citrusleaf, Membase, etc.

44

As previously discussed, KVP is the foundation upon which NoSQL databases are built. The image above illustrates common kinds of NoSQL databases – wide column, document, and graph data stores – with a brief description and some examples of each.

Exercise 3: Key-Value Pairs Modeling

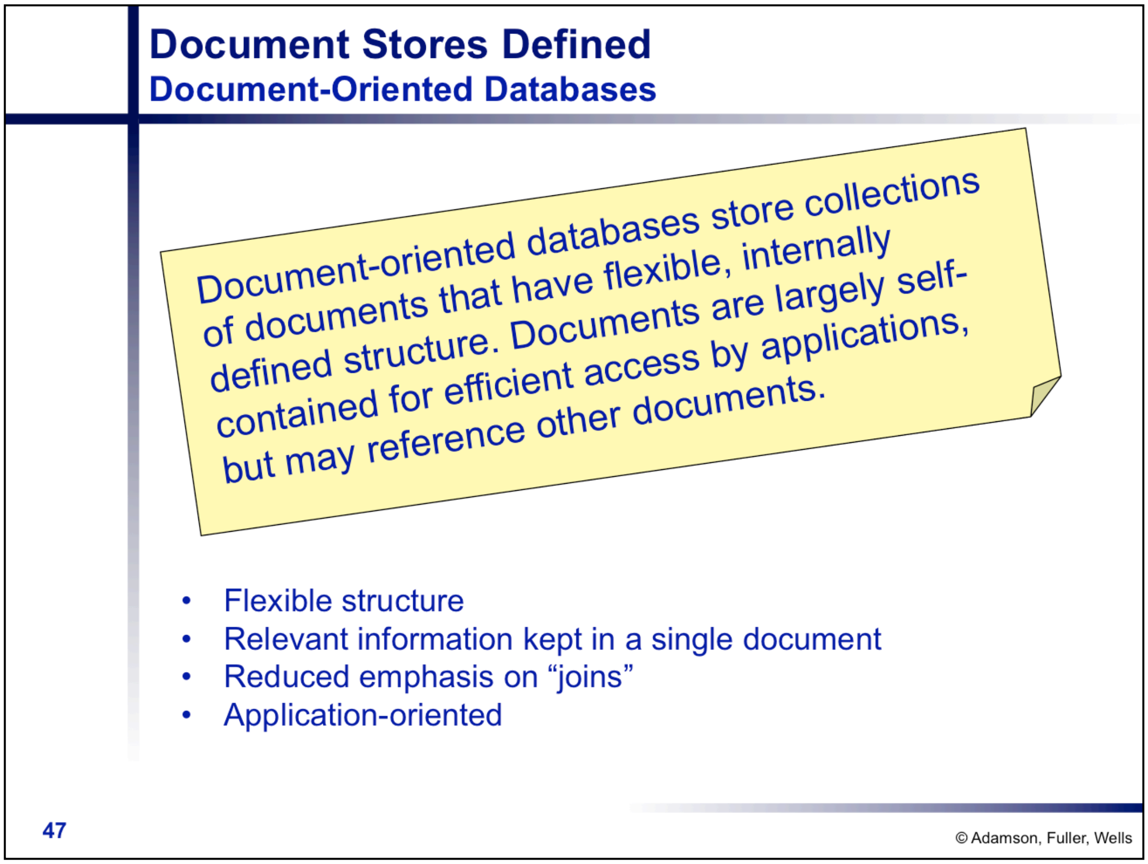
Warranty	4
PriusExterior	Beige
PriusTrim	III
Service Plan	Platinum
AccordExterior	Silver
Warranty	1
Service Plan	Gold
PriusExterior	Silver
PriusExterior	Charcoal
AccordInterior	Burgundy
PriusExterior	Green
AccordTrim	EX-L
AccordExterior	Black
AccordInterior	Blue
AccordExterior	Gray
AccordInterior	Gold
PriusInterior	Off White
AccordExterior	Green
AccordTrim	Hybrid
Warranty	2
PriusTrim	Two
PriusTrim	Five
PriusExterior	Red
AccordTrim	Touring
Warranty	out
AccordTrim	HybridTouring
AccordExterior	Red
AccordTrim	LX
PriusTrim	Three
AccordExterior	Gold
Warranty	6
AccordInterior	Black
Warranty	3
AccordExterior	Gold
Warranty	6

AccordInterior	Black
Warranty	3
PriusExterior	White
PriusInterior	Gray
PriusInterior	Black
PriusTrim	Plug-in
PriusTrim	II
PriusExterior	Gray
PriusTrim	Persona
AccordExterior	Beige
AccordInterior	Off White
AccordInterior	Gray
AccordTrim	Sport
AccordExterior	Burgundy
AccordExterior	Blue
PriusExterior	Black
PriusExterior	Turquoise
PriusInterior	Tan
PriusTrim	Four
PriusExterior	Blue
AccordInterior	Beige
AccordTrim	EX
PriusTrim	Touring
AccordTrim	LX-S
Warranty	5
AccordInterior	Tan
AccordTrim	SE
AccordExterior	White
ServicePlan	Bronze
PriusInterior	Beige
PriusTrim	IV
AccordTrim	Hybrid EX-L
ServicePlan	Silver
ServicePlan	Lifetime

Part 4

Document Stores

Document Stores Defined
Document Data Representation
Use Cases
Examples

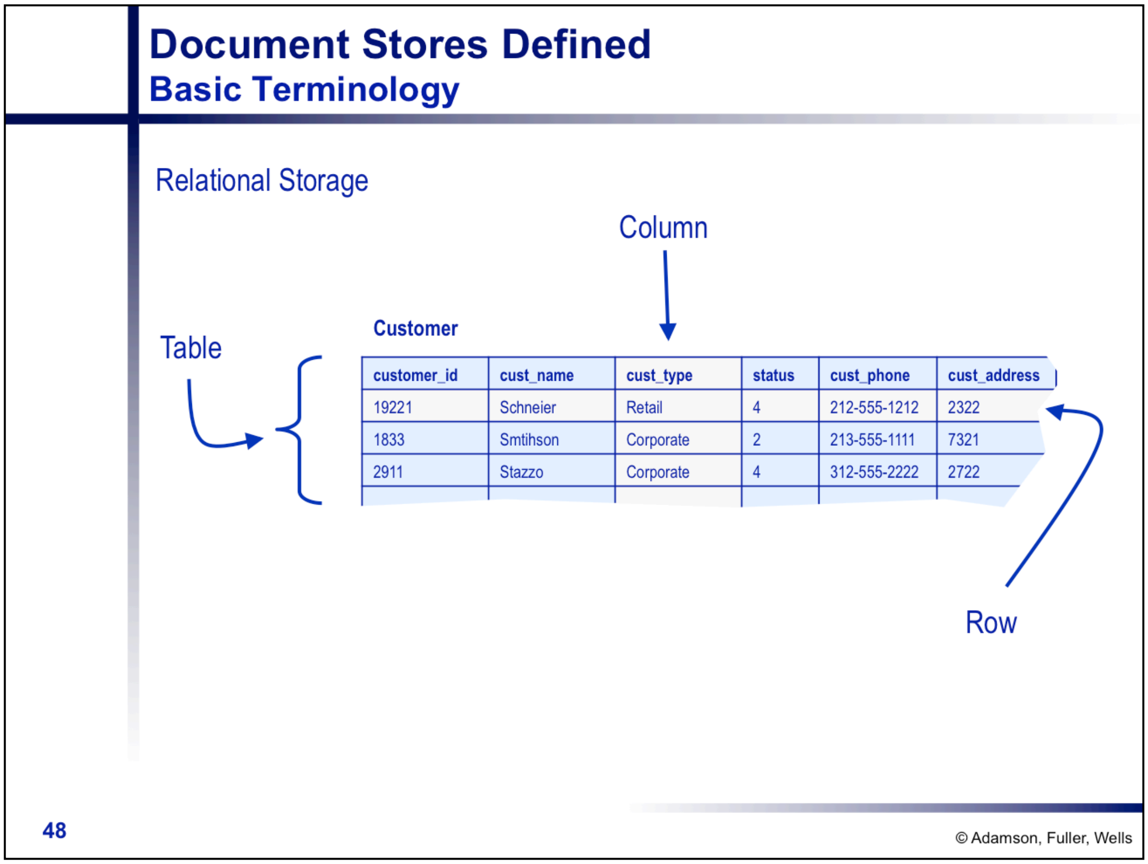


Document Databases (also referred to as **Document-oriented** Databases) manage self-contained units called….documents.

Documents do not have pre-determined structure. Instead, they have internal, self-defined structure.

Documents that describe a single business concept, like a customer, are referred to as a collection. The documents in a collection are not required to have the same structure.

Documents can also contain repeating attributes (called arrays) or even other documents. They may also refer to one another, but it is up to applications to be sure these associations are accurate.

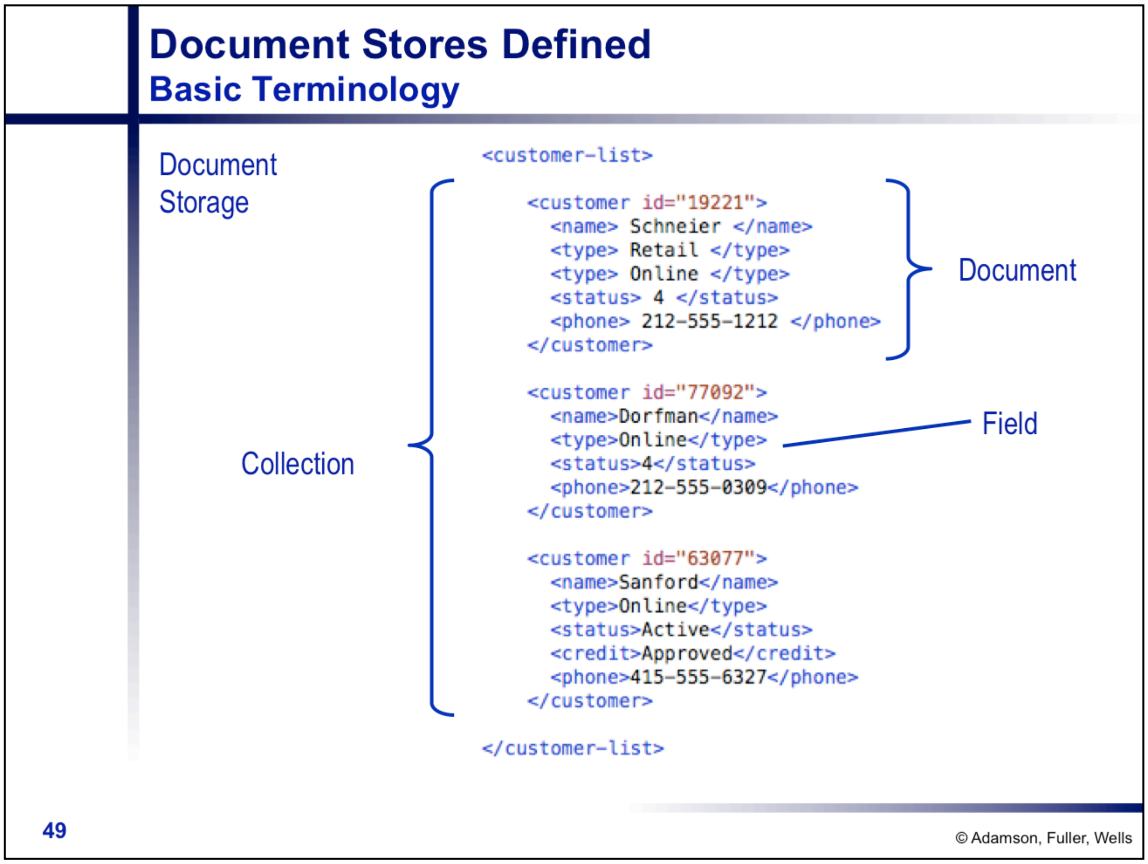


Database designs for relational data stores use the terms Table, Column and Row to describe basic features.

Table: A table collects instances of a particular type of thing. In this case, the table is collecting customers.

Column: The definition of a table includes the attributes of the thing being described. Each attribute to be collected is called a column.

Row: A record that is physically stored in the table is called a row. Each row is inserted into a table, and contains values for each column.

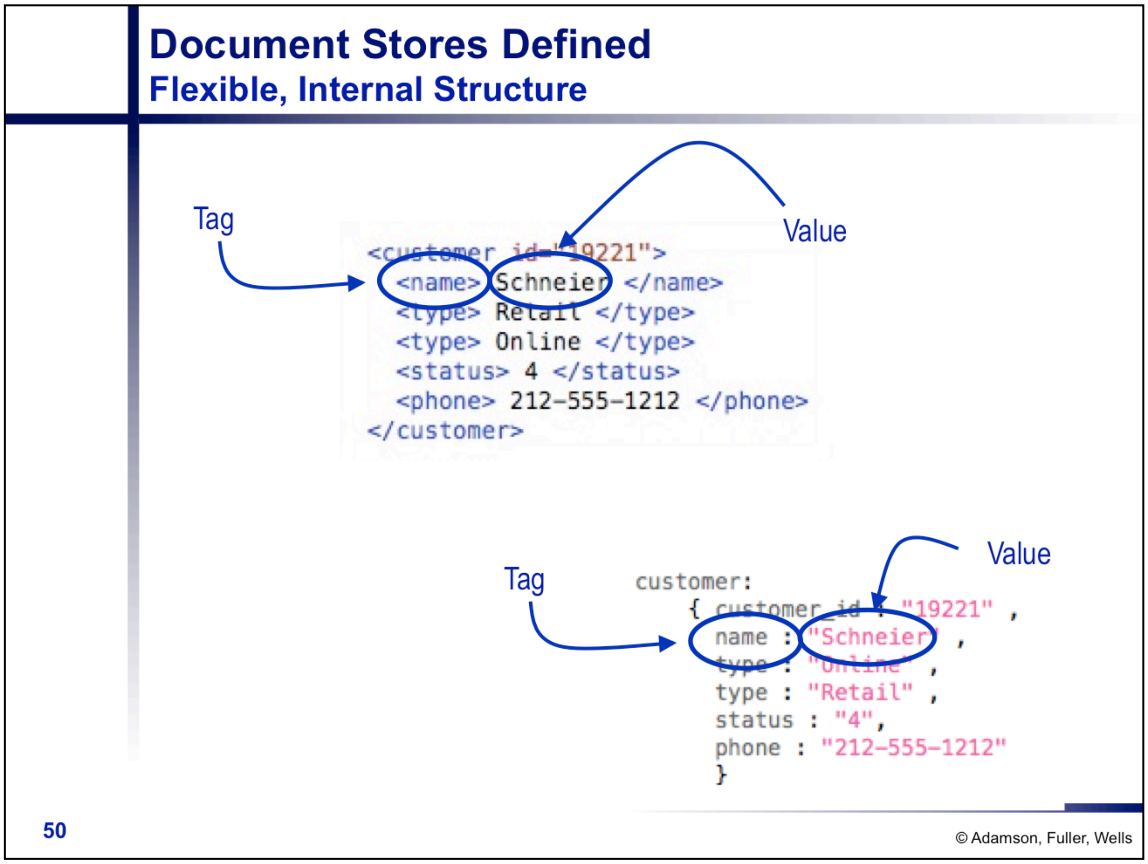


Document: A document store contains documents. Documents are like the rows of tables – they represent instances of things being described. (Documents also have some key differences from rows, as we will see in a moment.)

Field: Documents have fields. Fields are similar to columns in a relational store. (Unlike a relational store, though, the fields do not have to be defined in advance.)

Collection: Documents describing a single kind of thing are called a collection. Collections are like tables.

The primary focus of a document store is the document itself, which is similar to a row. Contrast this to a relational store, where the primary focus is the table. This nuance sheds light on some of the key differences between these approaches.



A collection does not define document structure. Instead, each document has an internal structure. This means that it is up to the applications that write documents to the repository to declare their structure.

Documents are typically formatted using XML (extensible markup notation) or JSON (JavaScript object notation.) These notations allow the declaration of both structure and content.

Documents are made up of fields, and fields have two parts: a tag describing the filed, and its value.

Document Stores Defined

Document Stores and Key-Value Stores

Key Value Pairs

```
<customer-list>

    <customer id="19221">
      <name> Schneier </name>
      <type> Retail </type>
      <type> Online </type>
      <status> 4 </status>
      <phone> 212-555-1212 </phone>
    </customer>

    <customer id="77092">
      <name>Dorfman</name>
      <type>Online</type>
      <status>4</status>
      <phone>212-555-0309</phone>
    </customer>

    <customer id="63077">
      <name>Sanford</name>
      <type>Online</type>
      <status>Active</status>
      <credit>Approved</credit>
      <phone>415-555-6327</phone>
    </customer>

</customer-list>
```

51

These fields are similar to key-value pairs. In fact, document oriented databases are often built on top of key-value stores.

Document Stores Defined
Fields Can Have Multiple Values

```
<customer id="19221">
  <name> Schneier </name>
  <type> Retail </type>
  <type> Online </type>
  <status> 4 </status>
  <phone> 212-555-1212 </phone>
</customer>
```

Multiple values (array)

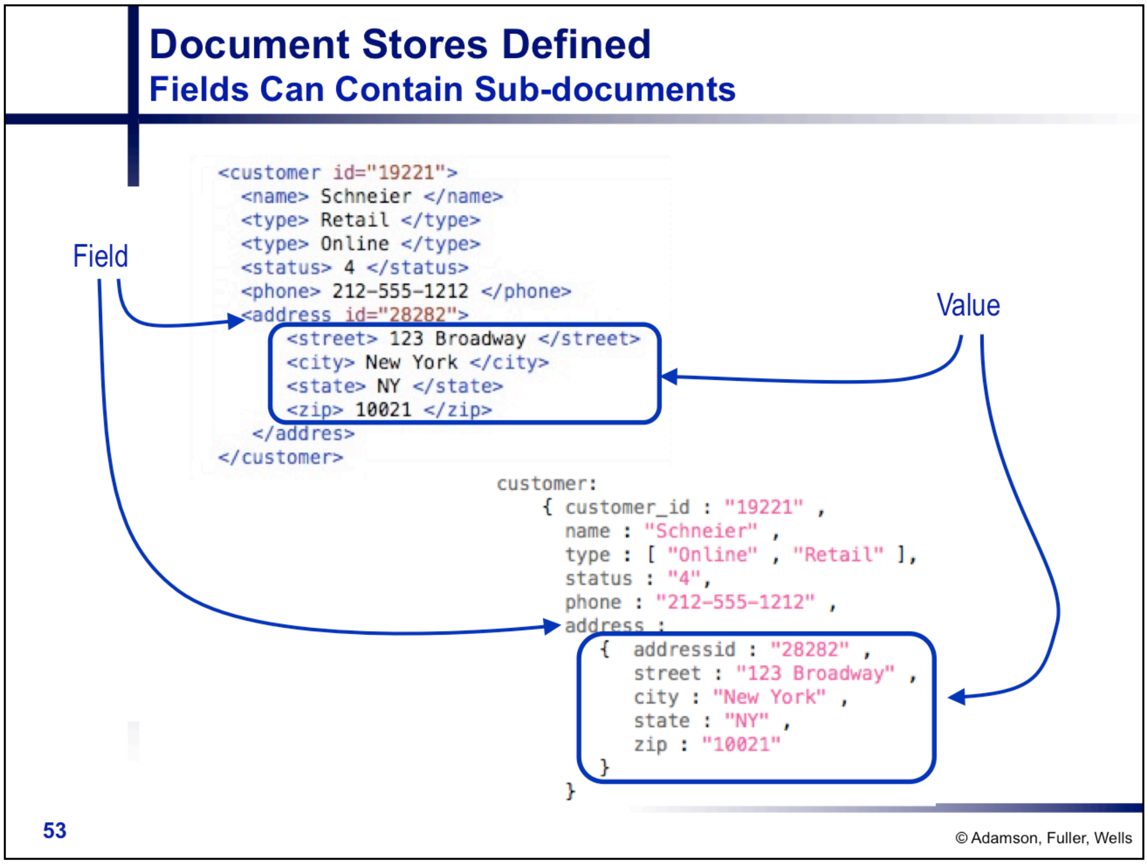
```
customer:
    { customer_id : "19221" ,
      name : "Schneier" ,
      type : [ "Online" , "Retail" ],
      status : "4",
      phone : "212-555-1212"
      }
```

52

Unlike a relational design, a field can have multiple values. In a relational model, this kind of data would likely require multiple tables.

For example, the document above describes a customer who has multiple customer types – retail and online. These two types are represented in the single customer document.

In an ER model, the repeating attribute would usually be placed in a separate table, with a many-to-many relationship to customer.



Documents can contain other documents.

In this example, a customer document contains a field called address. The address value itself has a structure like a document, with its own set of fields and values. It is usually referred to as a subdocument, since it is not stored separately.

Document Stores Defined
Summary of Characteristics

Document Store

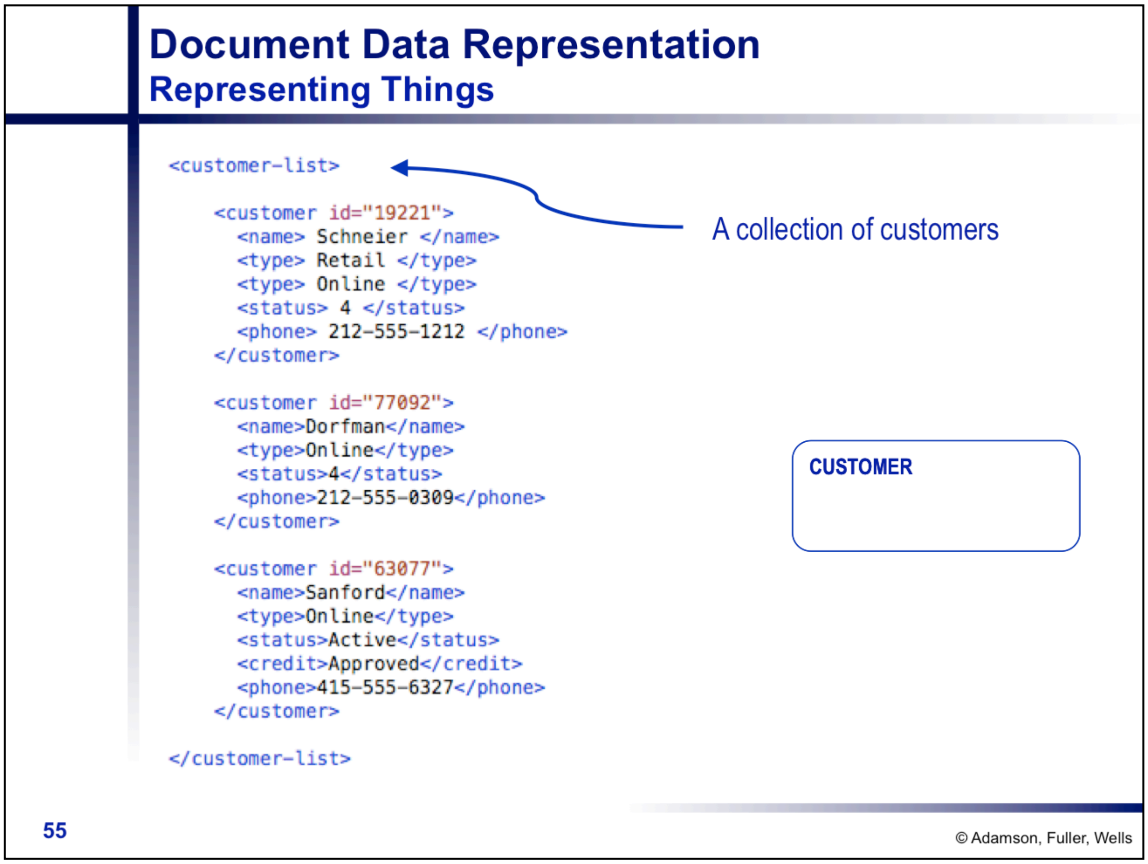
- Document structure is flexible and declared when created
- Documents define their own structure (not collections)
- Documents in a collection can have different structures
- A field can have different data types across documents
- A field can have multiple values in one document
- Documents can be nested

Relational Store

- Row structure is uniform and declared in advance
- Rows do not define their structure, the table does
- Rows in a table have the same structure
- A column has the same data type across rows
- A column has a single value for each row
- Rows cannot contain rows

Unlike a relational store:

- The structure of documents in a collection is not defined in advance. (In relational storage, the structure of each row is defined in advance.)
- Documents define their own structure; the collection does not control document structure. (In relational storage, the table defines the format of a row.)
- Document structure can vary from document to document. The fields do no not have to be identical. (In a relational table, each row has identical structure.)
- A field can have different data types across documents. (In relational table, a column has a single data type for all rows.)
- A field can have multiple values. (In relational table, each column contains one value per row.)
- Documents can be nested.



While document databases look very different from relational databases, they store the same kind of information. Using our standard notation, we can makes sense of what is contained in a collection of documents.

The top level organizing principle in a document database is the collection. In this picture, you see a collection of documents that describe customers.

Using our standard notation, we represent this collection as a **thing** called "Customer."

Document Data Representation
Representing Identifiers

```
<customer-list>

    <customer id="19221">
      <name> Schneier </name>
      <type> Retail </type>
      <type> Online </type>
      <status> 4 </status>
      <phone> 212-555-1212 </phone>
    </customer>

    <customer id="77092">
      <name>Dorfman</name>
      <type>Online</type>
      <status>4</status>
      <phone>212-555-0309</phone>
    </customer>

    <customer id="63077">
      <name>Sanford</name>
      <type>Online</type>
      <status>Active</status>
      <credit>Approved</credit>
      <phone>415-555-6327</phone>
    </customer>

</customer-list>
```

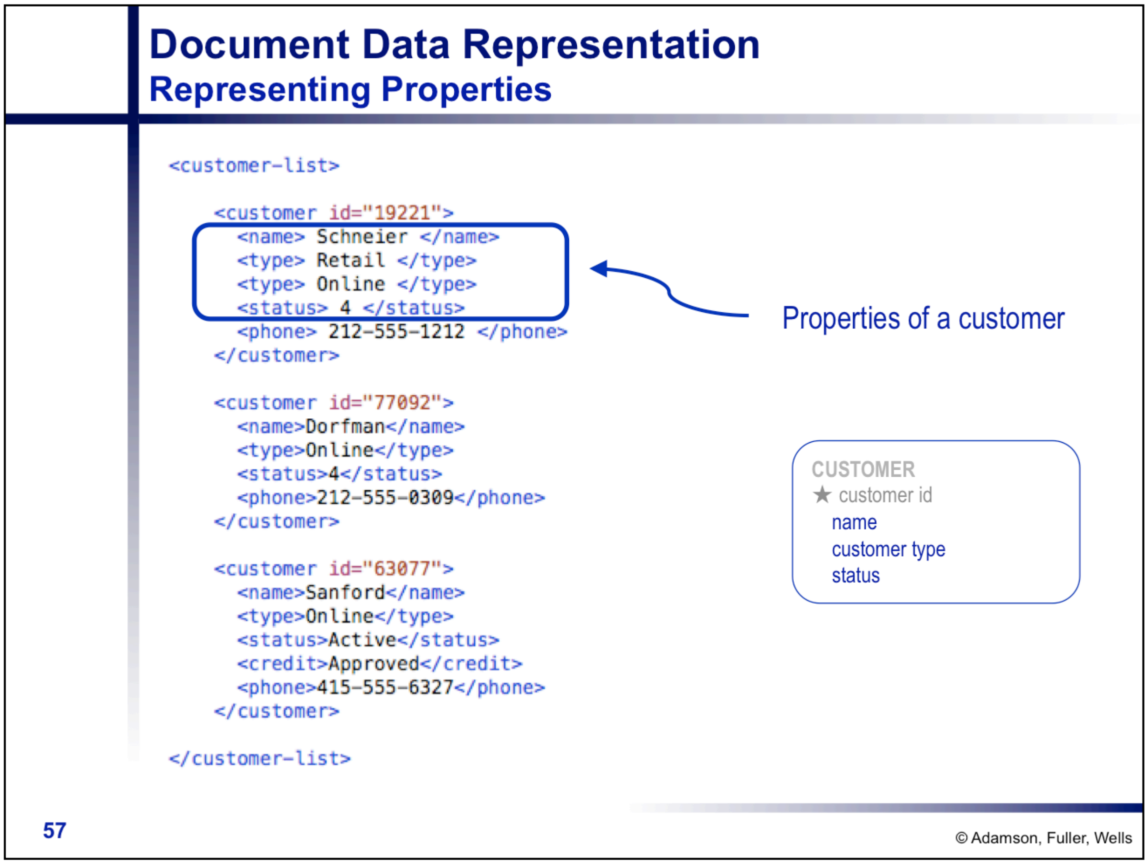
★ identifier of a customer

CUSTOMER
★customer id

56 © Adamson, Fuller, Wells

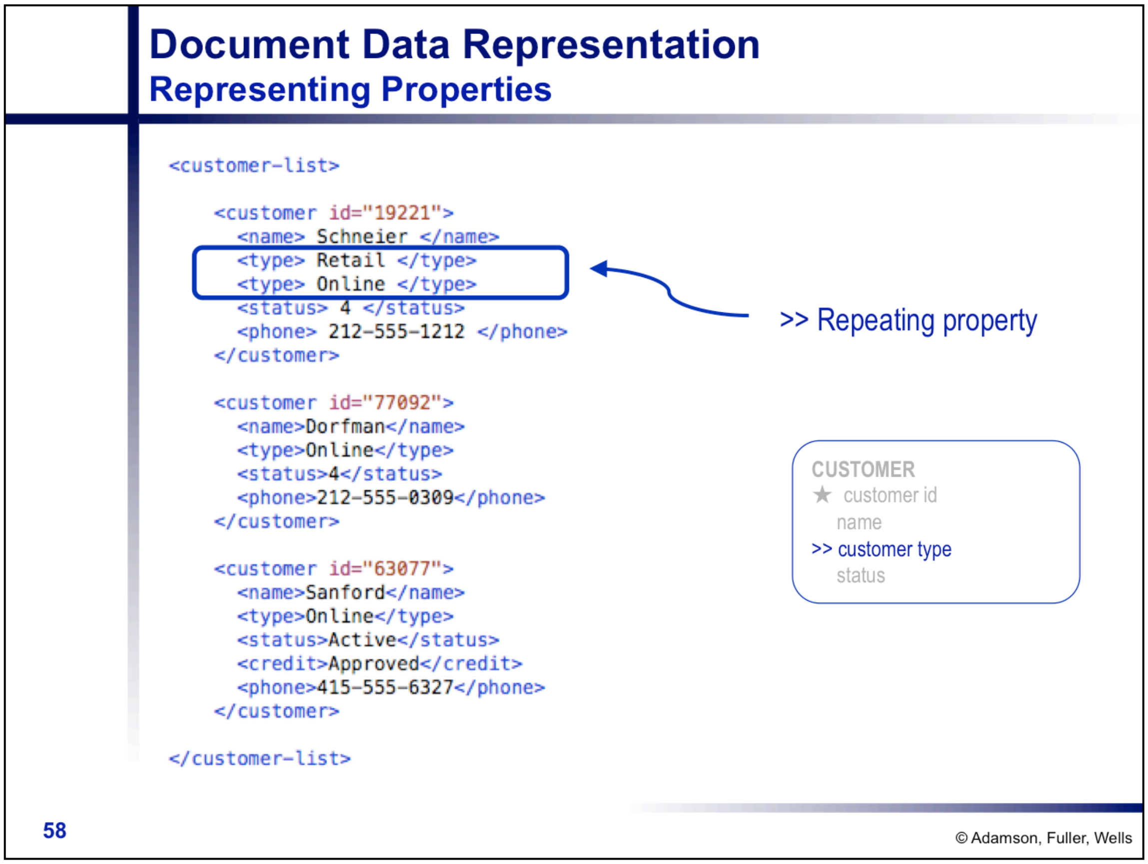
Each document within the collection has an ID, which serves as a key to access the document. Document ID's are identifying information.

Using our standard notation, the document id is listed with a star, indicating identifying information for a customer.



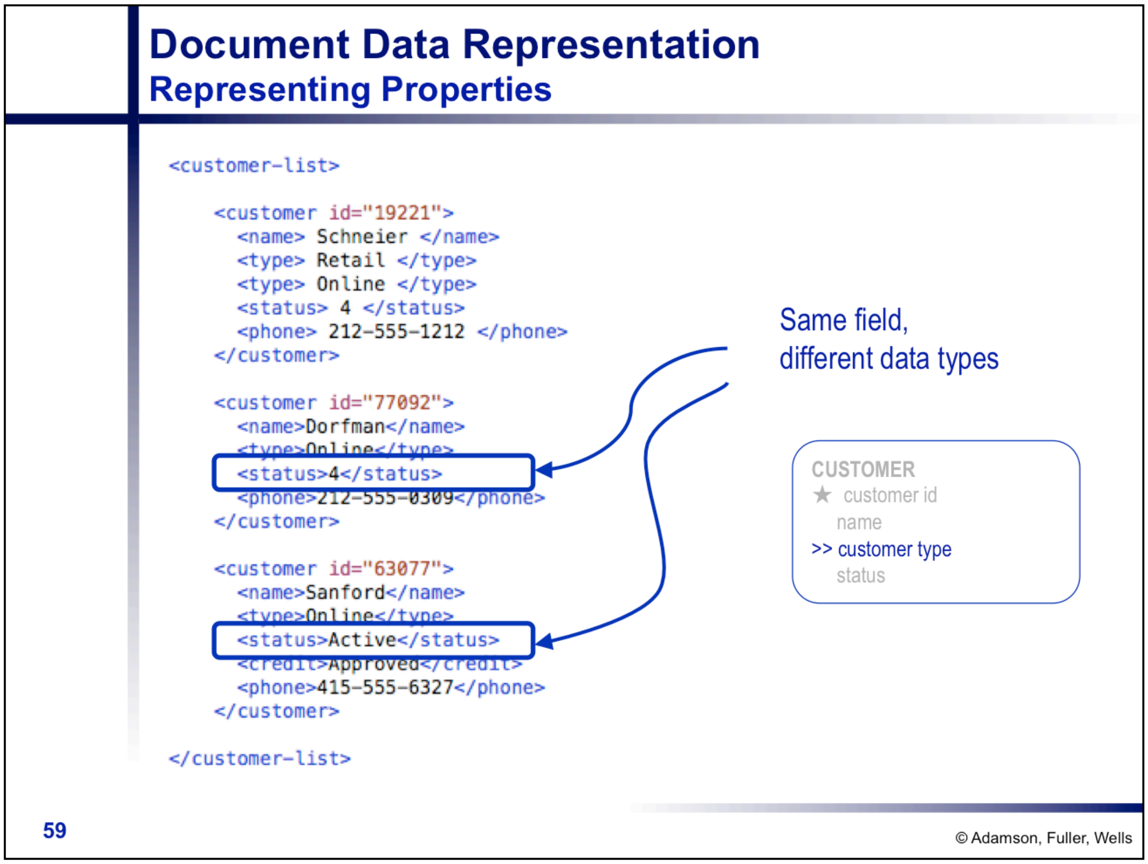
The various tags and values in a document provide information about the customer it describes. For example, the first document in this list contains Name, Type, and Status.

These key value pairs are **properties** of the **thing** called customer.



This customer has multiple types. In document parlance, it is an array.

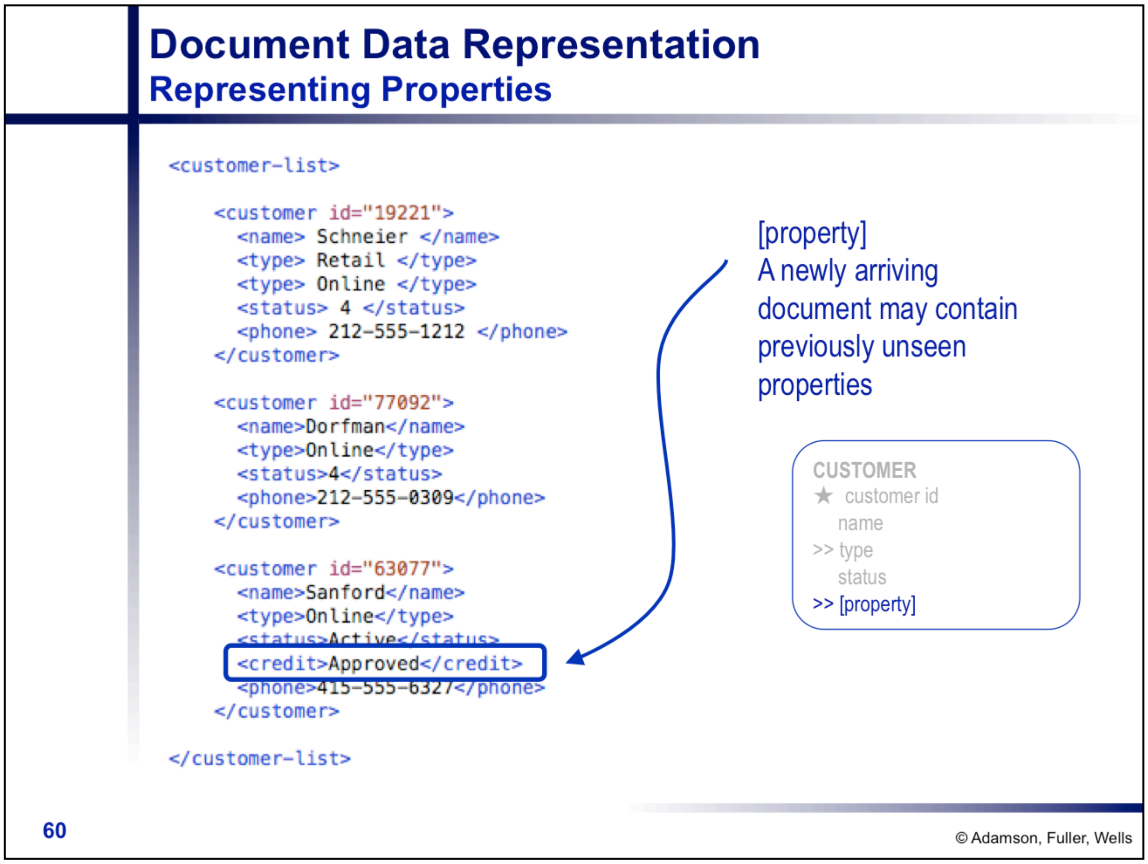
To indicate that a customer type can have multiple values, we place the repeating property symbol >> in front of it.



Notice that these two customers have a property called Status, but the values are integers in one case and a string in another case.

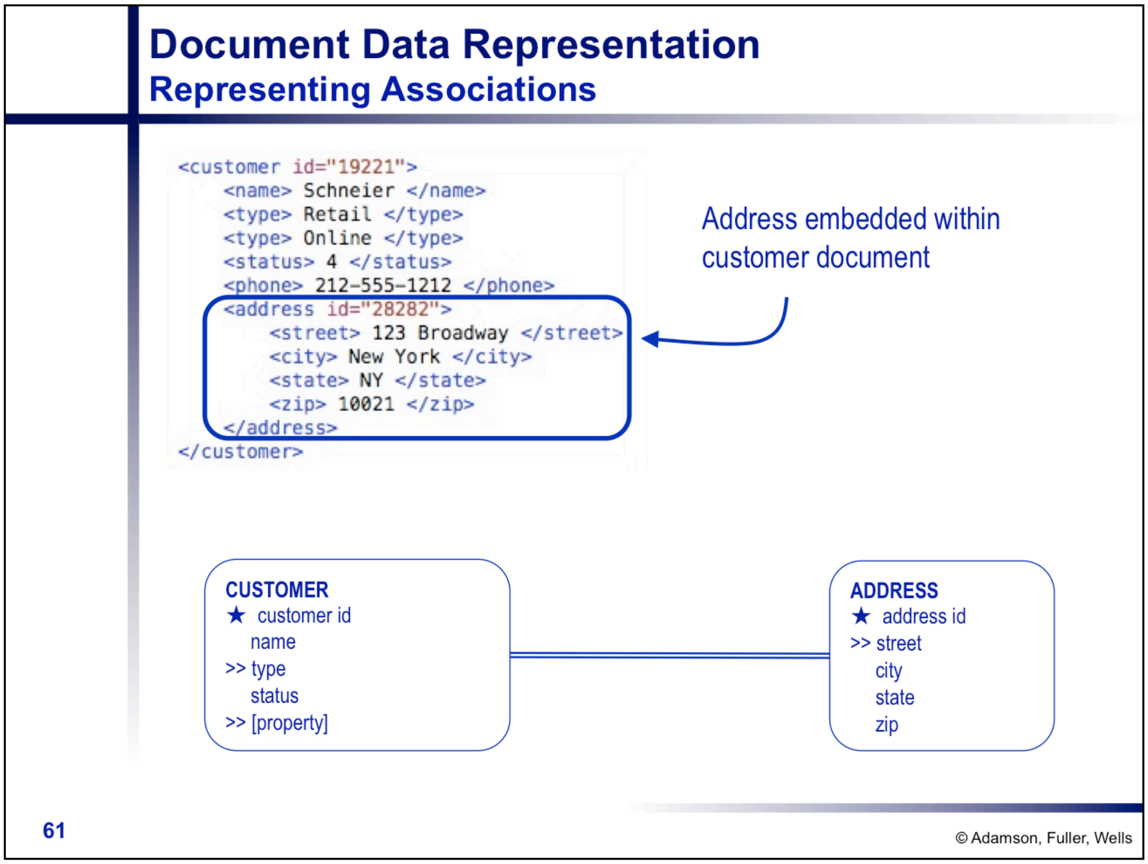
The notation describing the customer thing does not express data types. However, it will be useful to track the different values, and possibly map them to one another.

For example, status “4” may be equivalent to status “Active”. It will be necessary to know this when searching documents for active customers.



Documents do not have fixed structure. Each time a customer document is written to the repository, it may contain additional fields that have not been seen before.

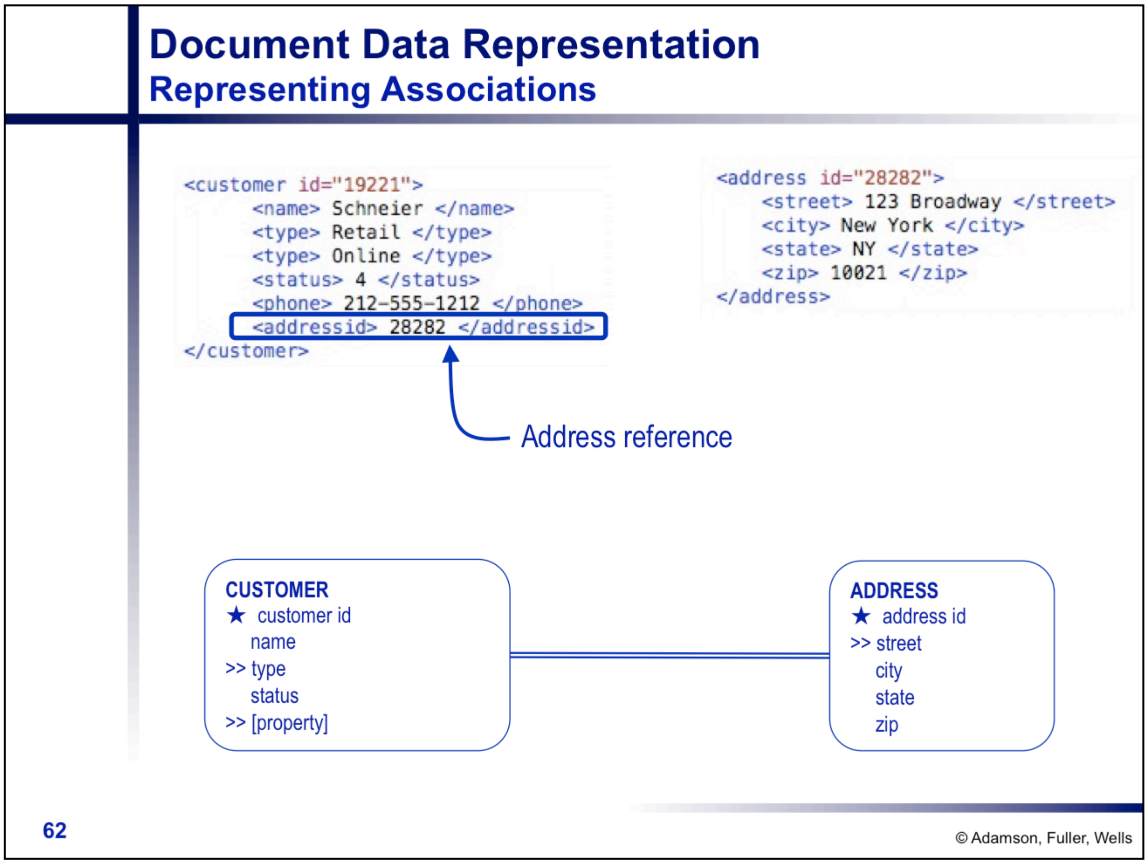
We can represent this flexible structure by declaring a generic property inside square braces. Since it is possible for a document to have multiple properties that were unanticipated, the repeating property symbol >> also appears.



In a document oriented database, an association can take two forms. The first is an embedded association.

Here, the address is contained as part of the same document that describes the customer. Embedding the address makes the document easier for an application to work with. Everything about the customer can be found in a single document.

This is the normal way to organize documents in a document store; documents are self encapsulated and include all necessary information about their subject.



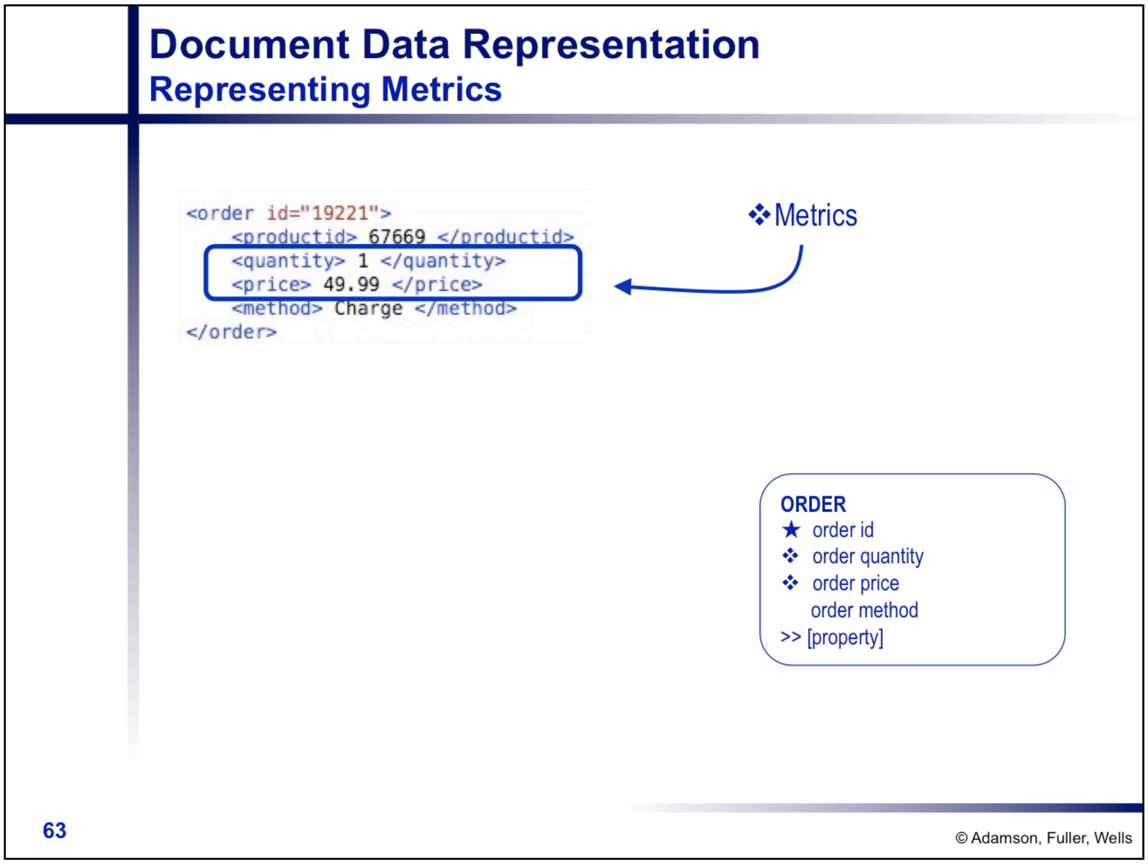
The second form of association is association by reference.

Here, the customer document does not contain the address. Instead, it contains an address ID. This is the identifier of a specific address in a collection of addresses.

In a document database, reference by association places additional burdens on the application. Among other things:

- The database does not guarantee that address ID 28282 exists in the address collection. It is up to application programmers to see to referential integrity.
- If an application needs a customer and their address, the application has additional work to do.

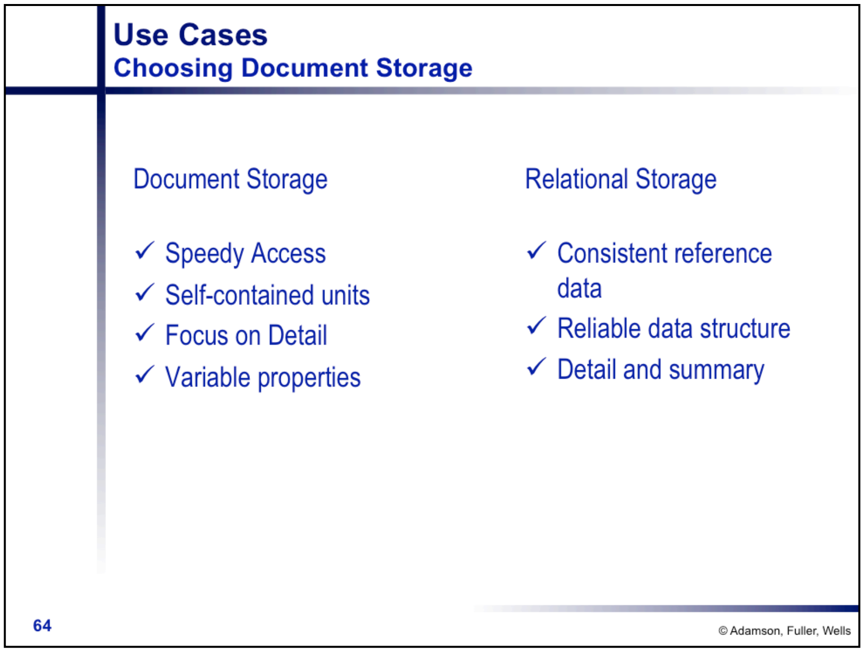
Normally, documents are organized to be self-contained, meeting the needs of applications without requiring additional programming for “joins.”



Documents do not distinguish properties from metrics, but from a business perspective we do.

This document, for example, contains fields that describe order quantity and order price. These fields are metrics; we can imagine slicing, dicing and aggregating them.

In our representation of an order, they are flagged with the metric symbol.



Document storage is convenient when applications will focus on self contained units, such as customers, orders, log records, and so forth. While it is possible to request a group of related documents from a document store, this is more complicated form an application development standpoint.

Document stores do not enforce referential integrity within or across collections. If you have an order document that contains a product ID, the repository will not be sure the corresponding product is up to date.

Document stores allow for flexible, variable nature of documents.

These characteristics have advantages when recording data, but can be difficult when querying data. The application looking for blue products must know to check for documents where product_color="Blue" as well as where is_blue="True". If a new document is added with the field is_blue="Yes", the application will not count it, nor will it count one that is inserted with the field exterior="Blue".

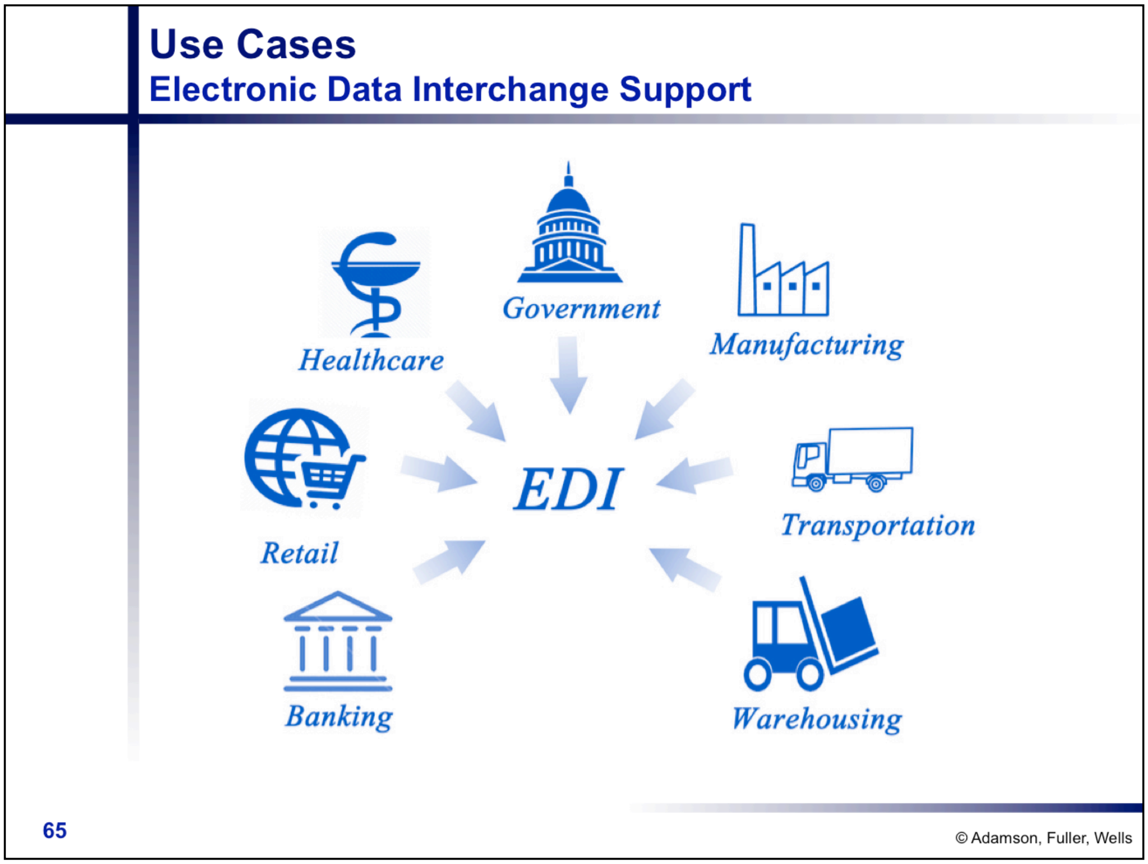
In the world of BI:

Capture: Document stores are suitable for intake, where the format of the source data is not controlled, or where there are multiple sources that do not adhere to a single source.

Explore: Document stores are suitable for business analytics, where the focus is on individual examples of things in the context of a single business activity.

Augment: Document stores are not suitable as the home of OLAP data, as they contain unpredictable fields, lack referential integrity, and are not well suited to aggregation. Instead, the document store may serve as a source to move important attributes into the structured world of the data warehouse.

Extend: Documents may be the format of record for details around a transaction. In this respect, a document repository may be linked to the data warehouse by way of



The document format and access methods are a natural fit in supporting electronic data interchange (EDI) between businesses.

For example, parts manufacturers may specify their product characteristics in an XML format. This information can be stored in a document database on either end of the exchange. The self-contained and self-defining characteristics of documents adapt nicely to the nature of many EDI formats.

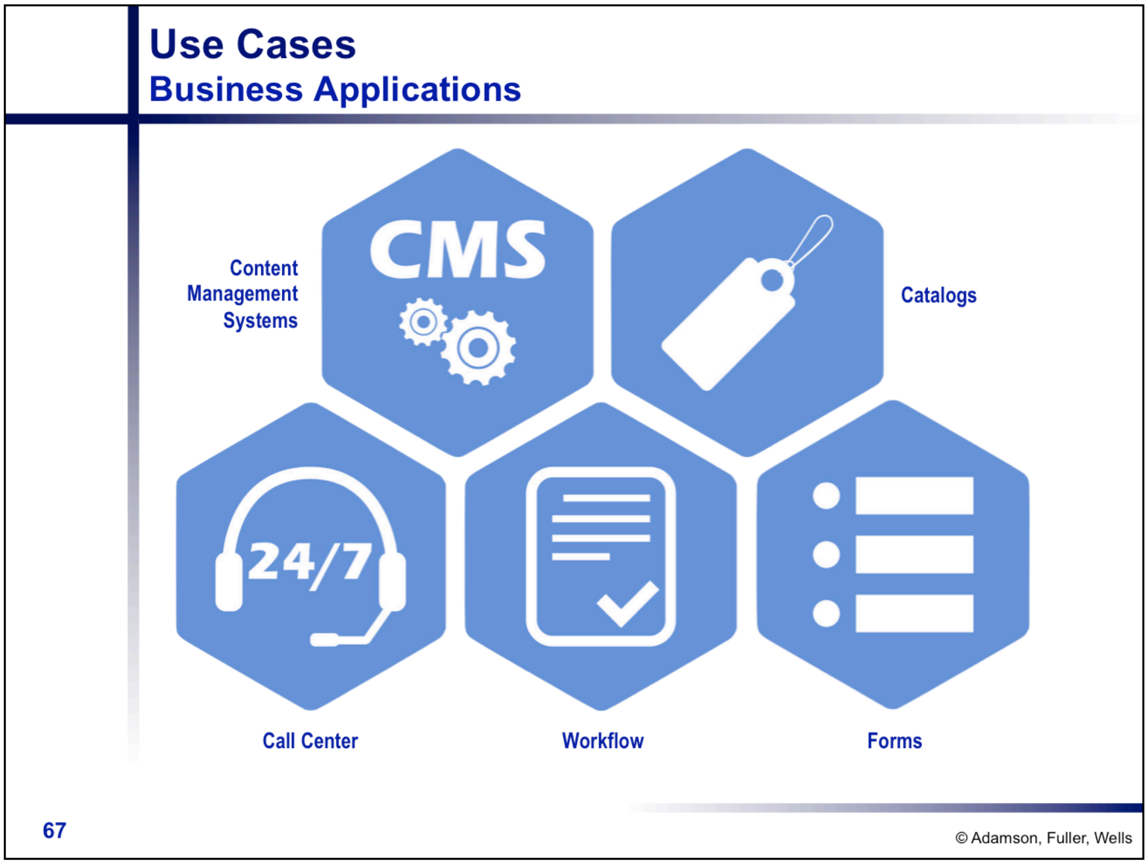
Use Cases

System Logs, Machine Generated Data

Machines and systems often store data in a document format.

Examples include:

- Logs
- Preferences
- Settings
- Alerts
- Notifications
- Activity results



The document format is used in a variety of business applications.

Content Management Systems use a document format to store the content of a web site, blog, or storefront. Formatting can be applied separately and tied to document characteristics such as Title, Header, Comment, etc.

Catalogs of products, services or resources exhibit a wide variation in attributes. The flexible nature of a document lends itself to specifying items of interest in a manner that may be more efficient than an RDBMS.

Customer Support applications can track incidents and related activities using a document format.

Workflow Management applications can specify documents (such as applications or claims), and activities (such as processing or adjudication steps), using a document format.

Forms used inside or outside of a business can be defined and filled in using a document database.

Part 5

Graph Databases

Graph Databases Defined
Data Representation
Use Cases
Examples

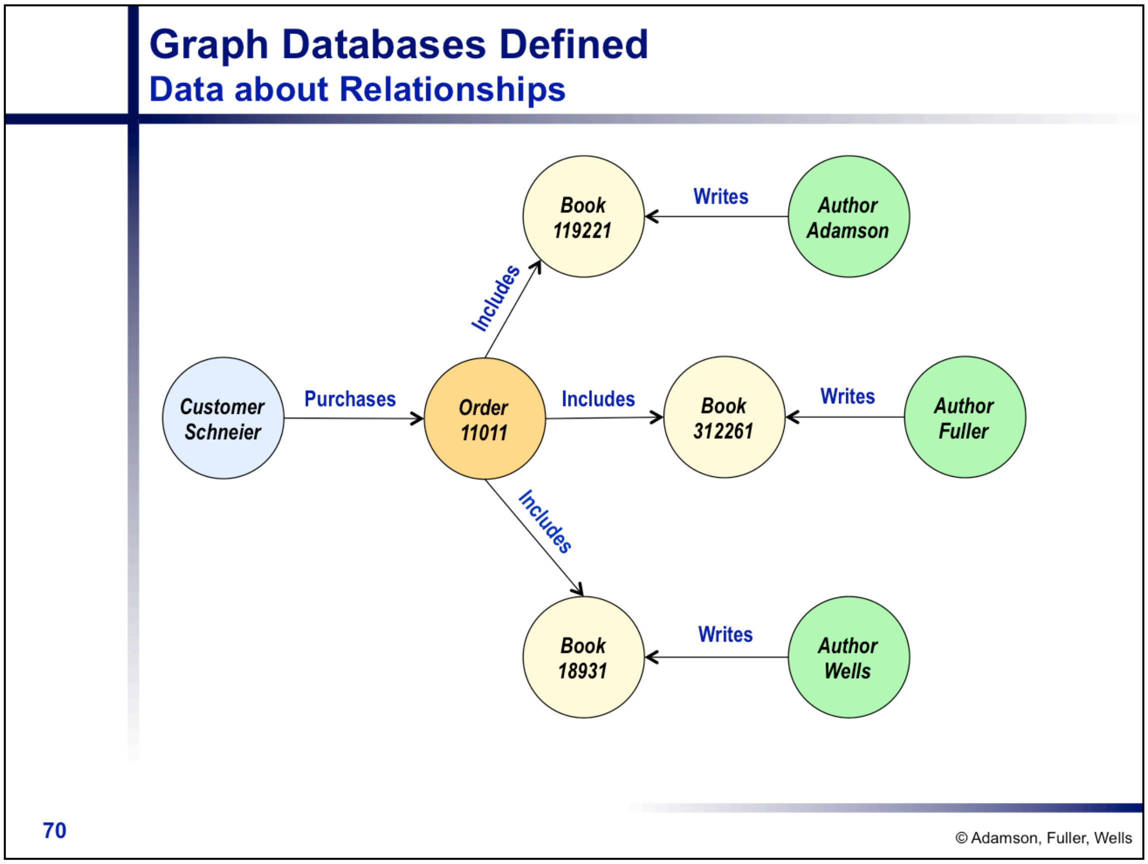
Graph Databases Defined
The Basics

A graph database is a database whose purpose is storage and access of graph data structures. A graph in this context is a map of relationships among things. A graph database connects the things that data represents without using indexes.

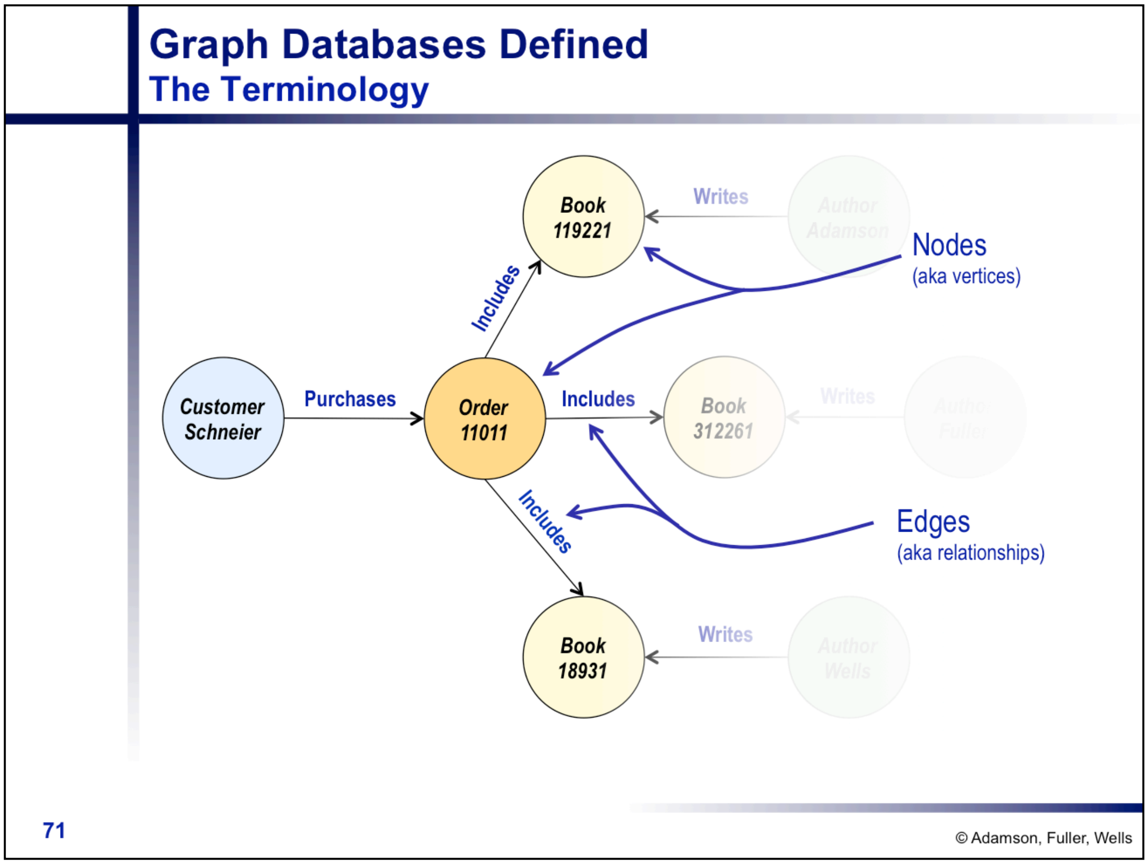
- Graph databases focus on relationships among things.
- Graph databases use direct pointers instead of indexing.
- Graph databases support create, read, update, and delete (CRUD) functions.

69

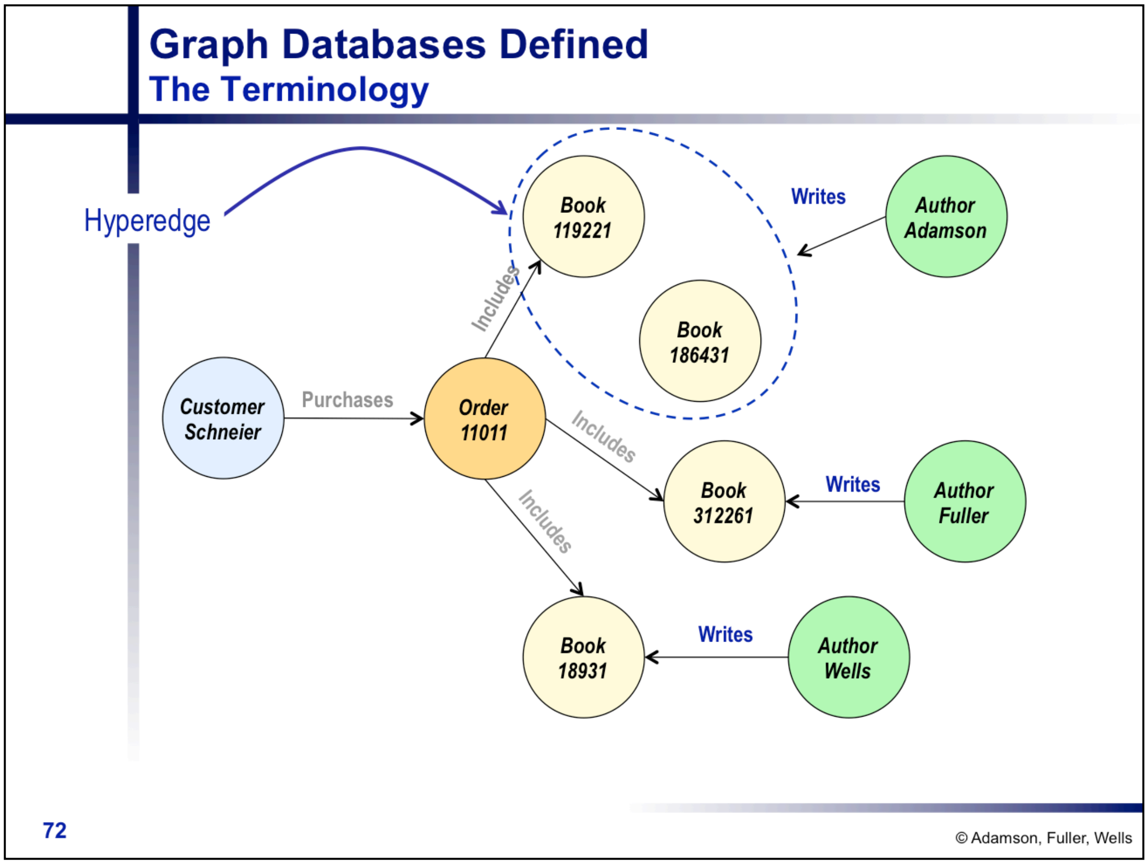
The purpose of graph databases is to describe relationships among things. The focus is distinctly different from relational databases where relationships are secondary to things. In a graph database the relationships are the primary data of interest and the many use cases are predicated on exploring relationships.



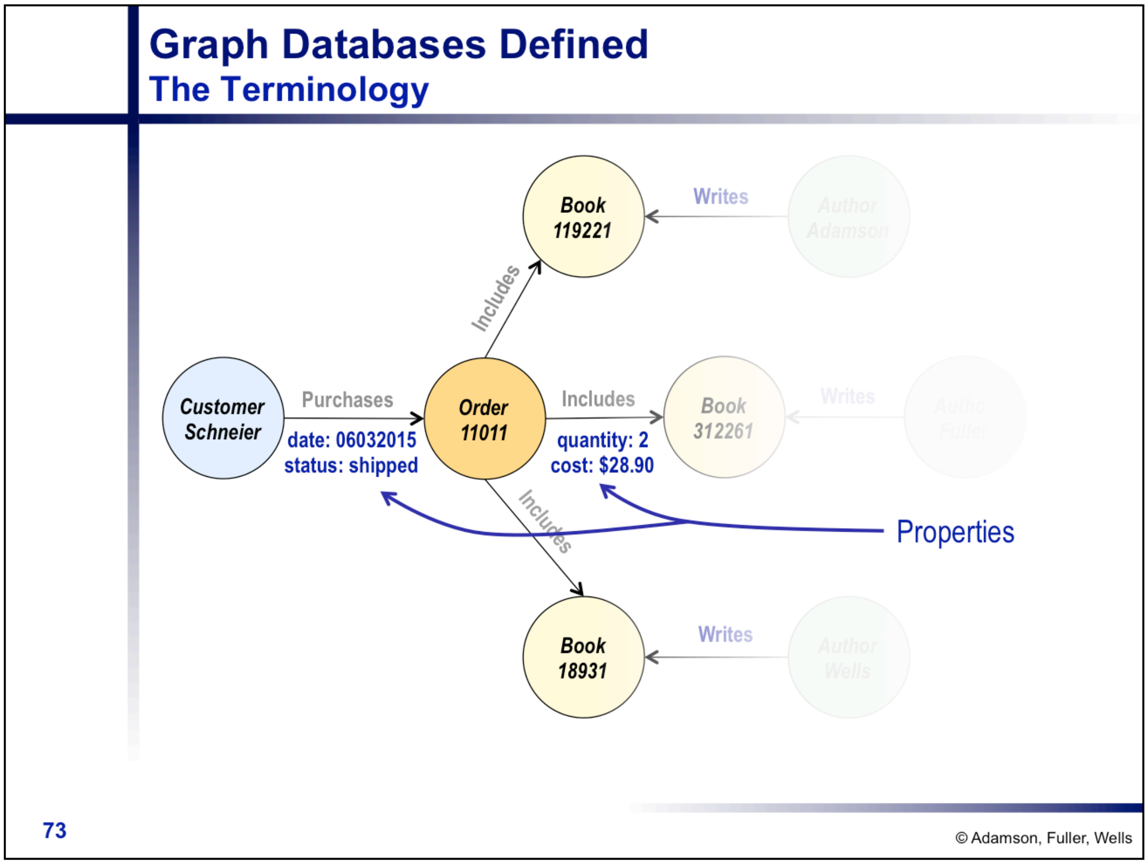
This example, seen earlier in the course, illustrates relationships of customers, orders, books, and authors. Note that relationships in this model are unidirectional – e.g. *customer purchases book.* In a relational implementation we would have a bidirectional – *customer purchase book* and *book purchased by customer* – with rules of cardinality for the relationship. The modeler's perspective on relationships changes when shifting from relational to graph databases.



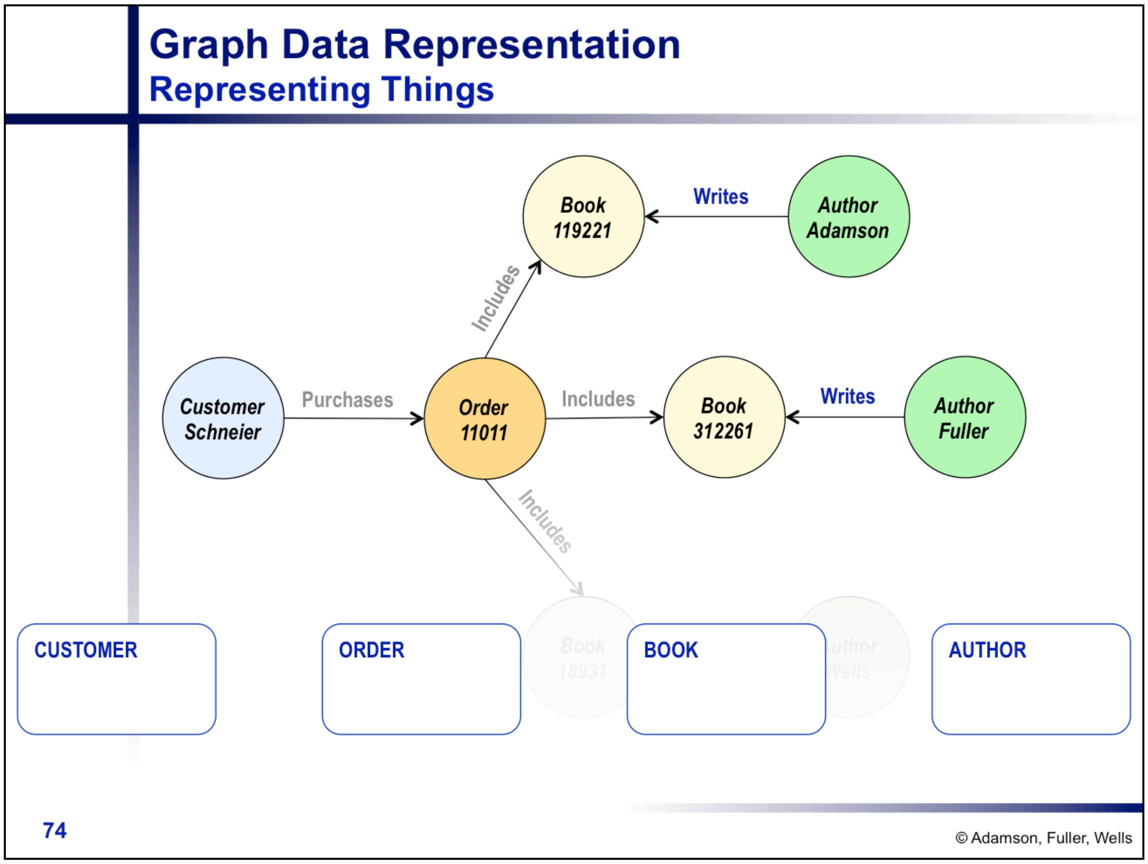
The language of graph databases is different from that of relational too. The things in a graph database are known as nodes or vertices. Relationships are called edges. Each node represents a thing (an entity in relational language) and each edge represents a relationship between exactly two nodes.



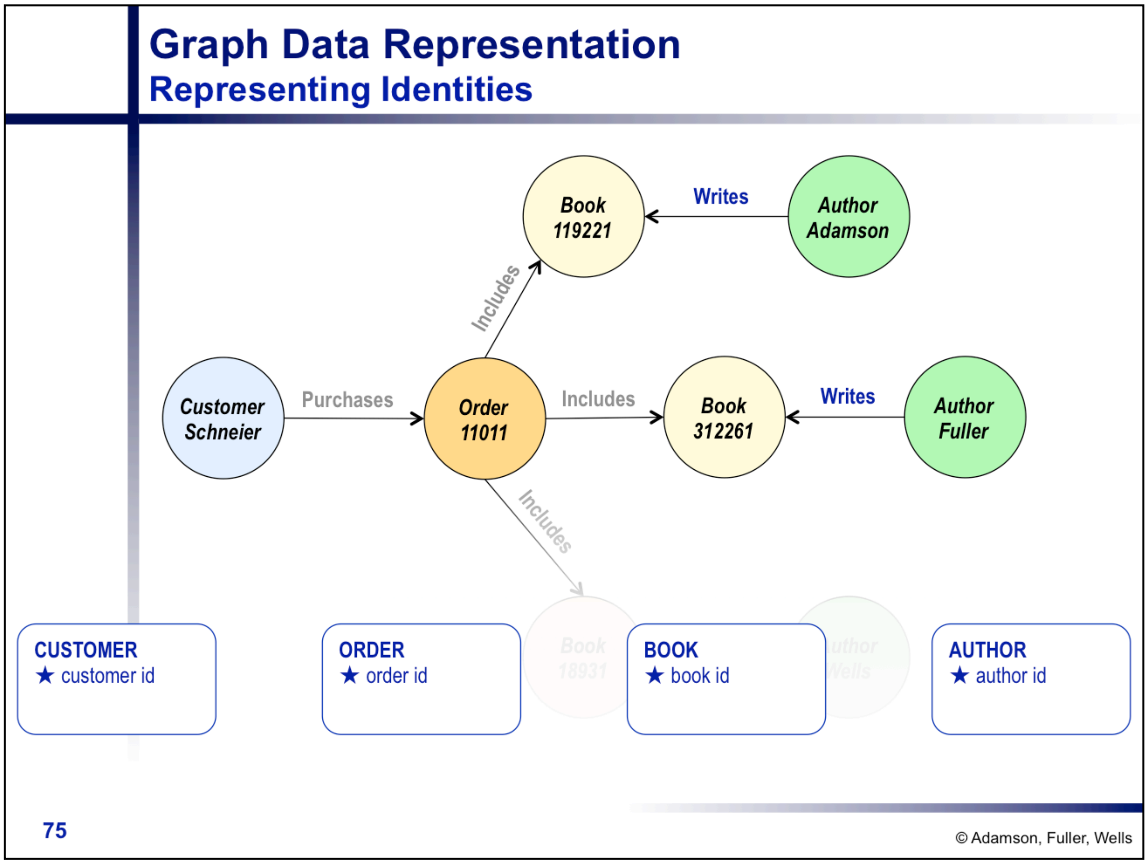
Some graph databases support the concept of hyperedges that allow more complex relationships than the binary relationship of exactly two nodes. A hyperedge is the boundary for a group of nodes using a set-theory concept – for example, all of the books by Adamson as a set. The hyperedge describes a the relationship of the group of nodes with other nodes, thus author Adamson is related to multiple book nodes not as several discrete edges, but as a single edge.



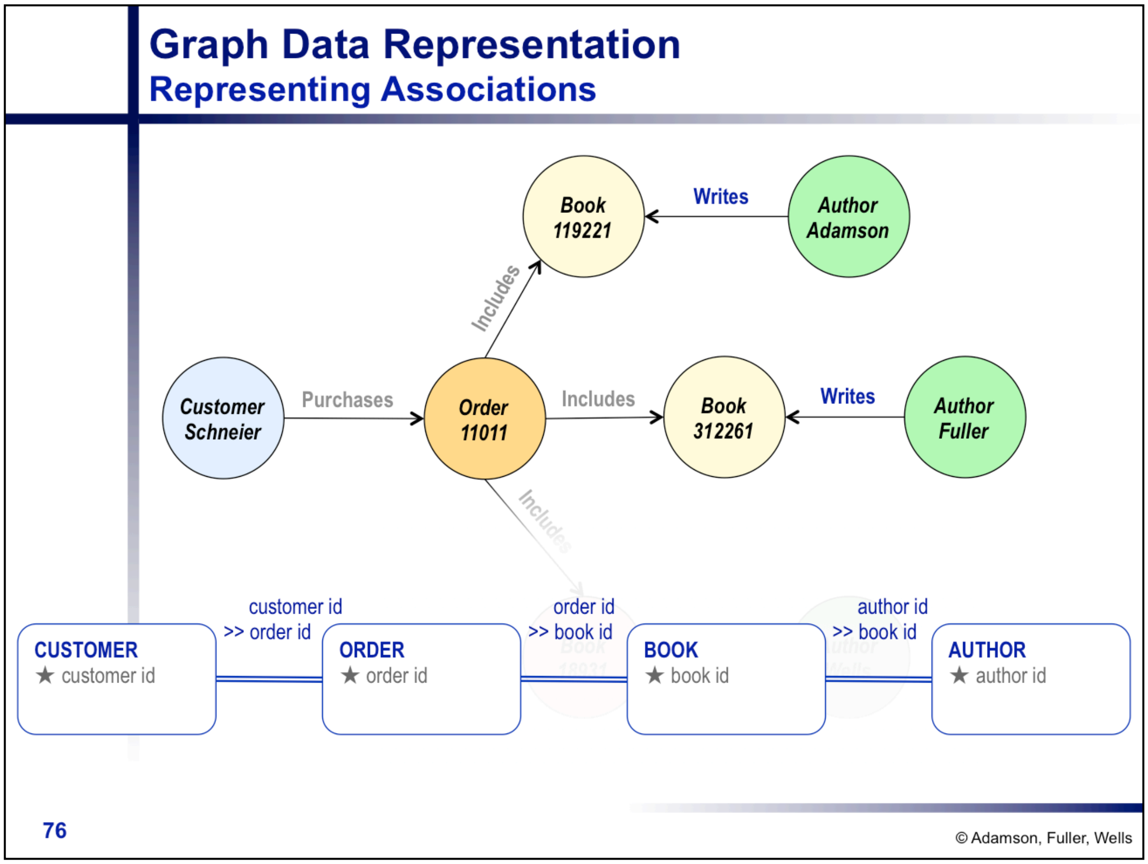
With the strong focus on relationships in graph databases it is typical that many properties are attached to edges. Unlike relational databases where entities have many properties and relationships have few, a graph database supports a large number of relationship properties.



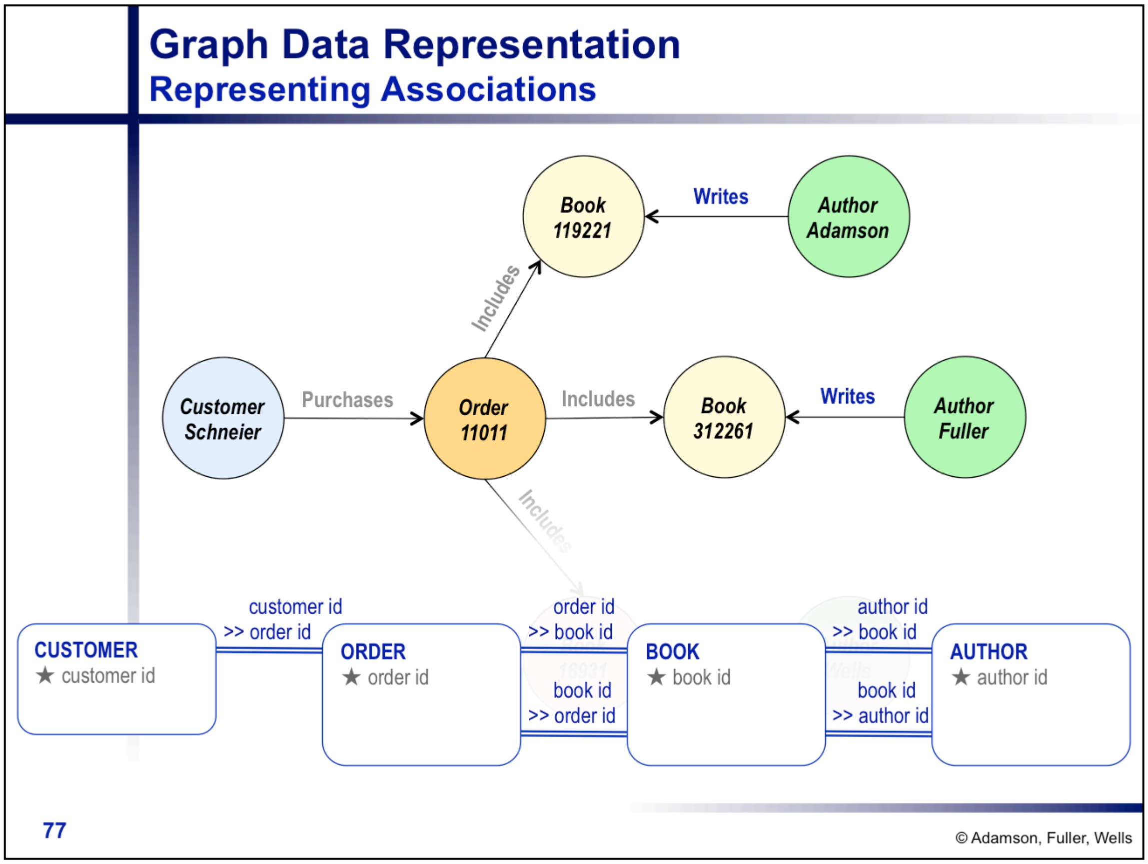
This diagram illustrates the modeling notation to represent things in a graph database model. Things correspond to nodes in the database structure.



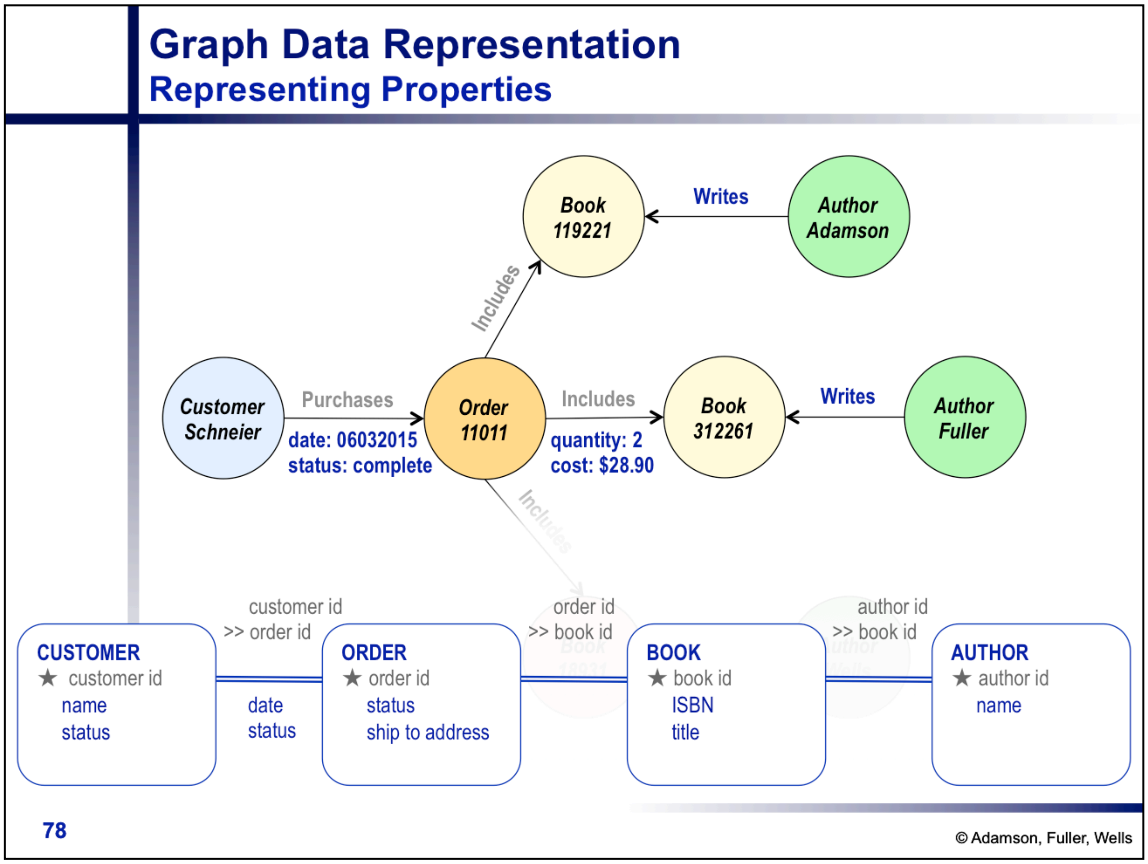
This diagram shows how identities are represented in the modeling notation. Things (nodes) have identity and identifying properties. Edges do not have identity distinct from the nodes for which they describe a relationship.



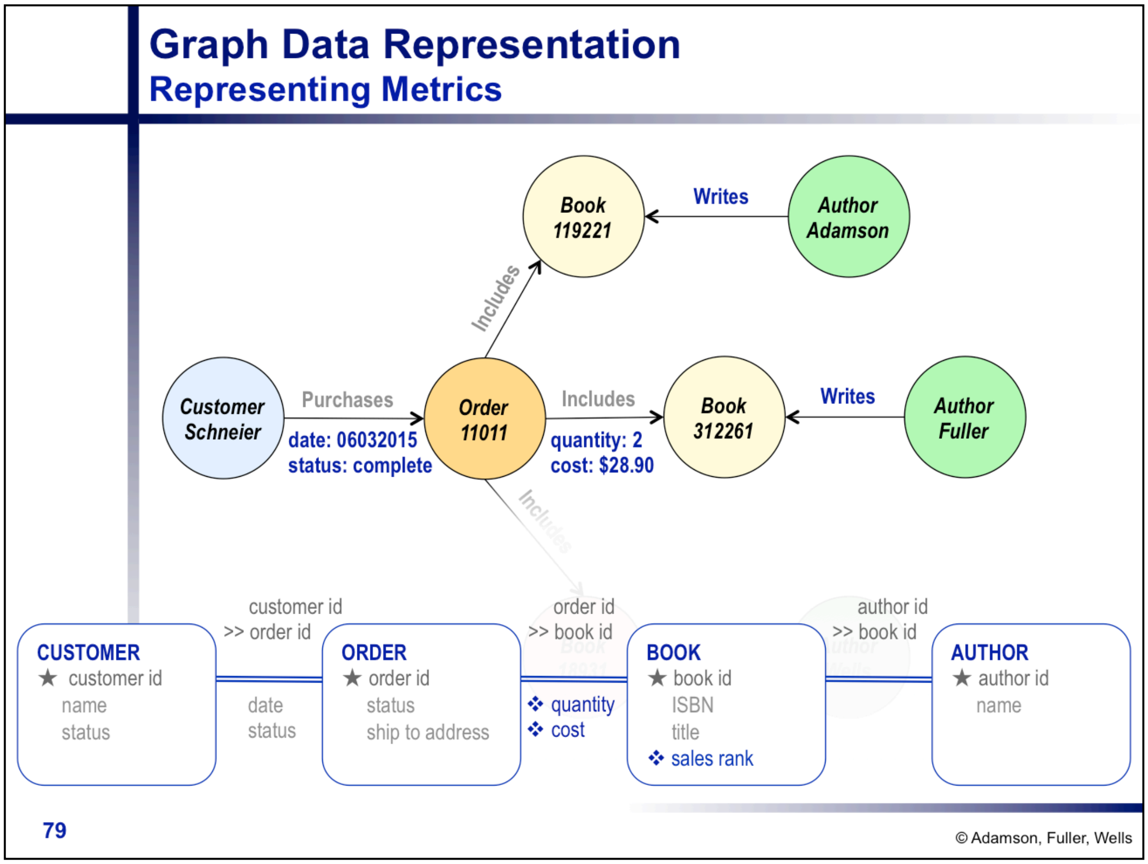
Edges are the associations in a graph database. The diagram above shows how they are represented using the modeling notation. Note the repeating identifiers >>*order id* and >>*book id*. The repeating characteristic is indicative of a one-to-many association – as close to the relational concept of cardinality as is practical in a graph database.



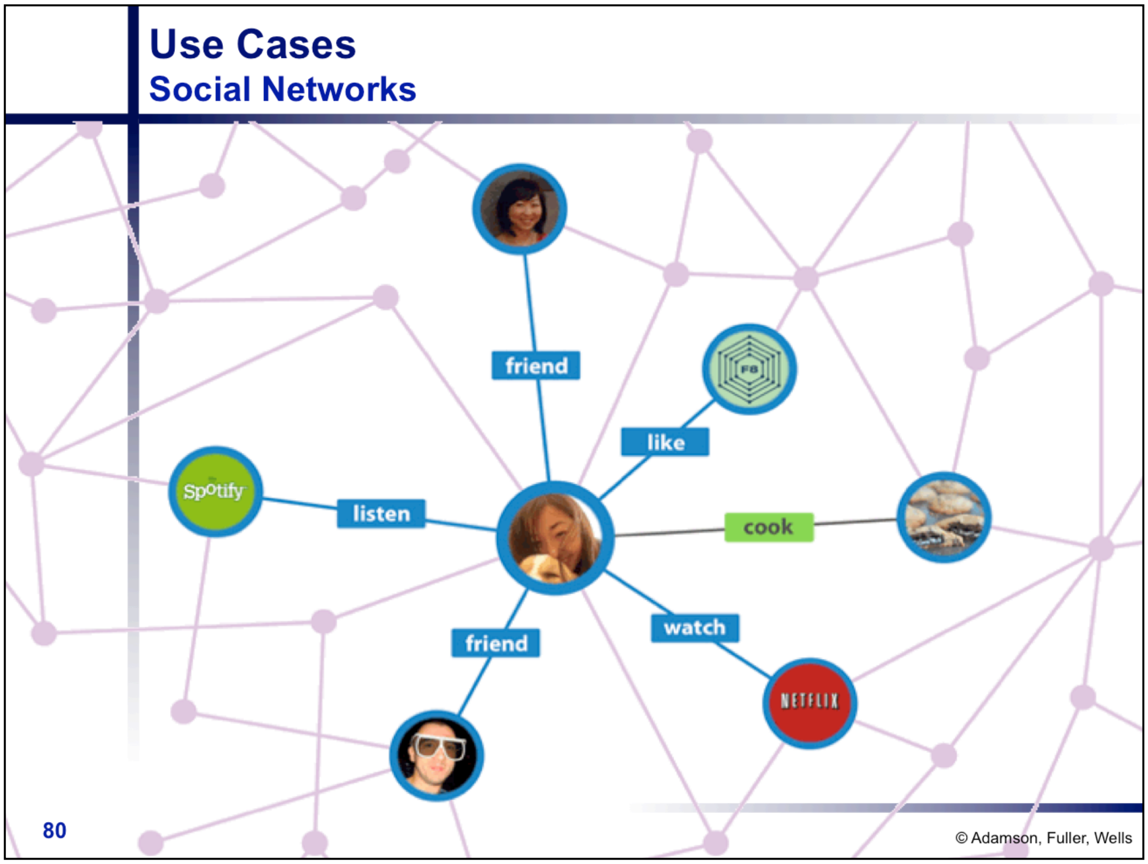
Many-to-many relationships do exist in the real world and are readily represented with entity relationship models, but in graph databases they are represented as two distinct one-to-many associations. The model example above illustrates that notation. One order is associated with many books (order id, >> book id) and one book is associated with many orders (book id, >> order id).



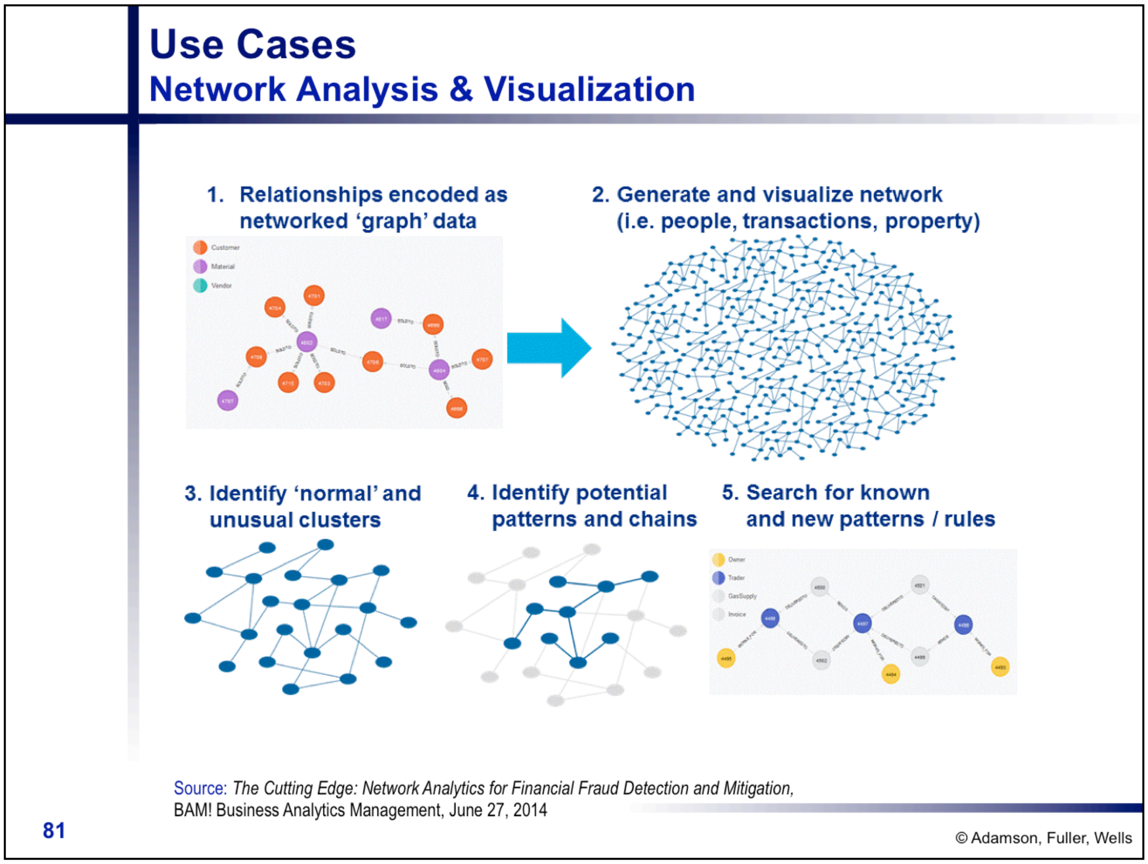
The diagram above illustrates representation of properties in a graph database using the modeling notation. Note that both nodes (things) and edges (associations) may have properties, although for many applications the main properties of interest describe the edges.



The diagram above illustrates metrics notation for a graph database. Any quantitative property (numeric and suited to mathematical manipulation) in a graph may be considered as metric data. Both nodes and edges may have metric properties. The example above shows the node *book* with a quantitative property of *sales rank.* It also shows metric properties – *quantity* and *cost* – for the edge associating order with book.

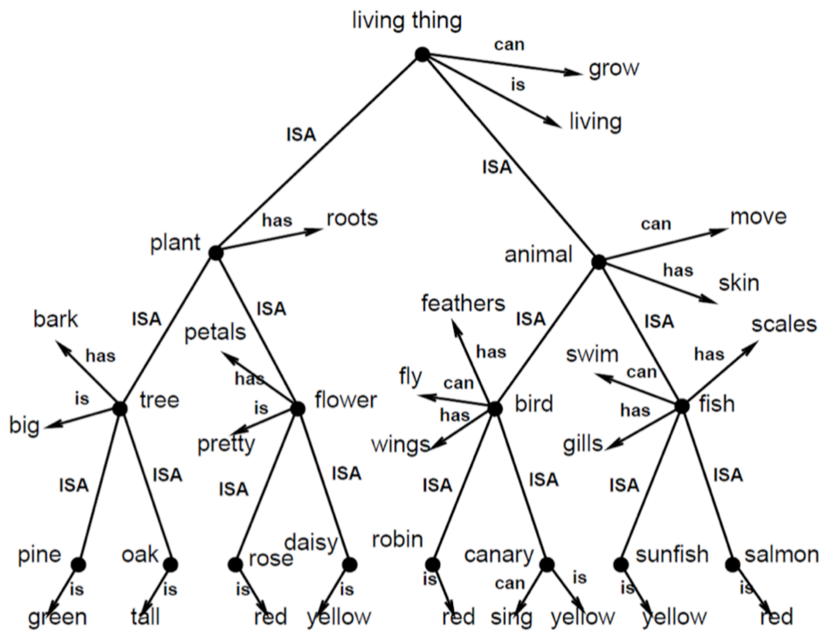


Social networks such as Facebook or LinkedIn connections are among the most common examples of graph database applications.



Visualizing networks of many kinds – people, transactions, property, etc. – is used to view relationship patterns. Understanding normal and exceptional patterns is valuable for applications such as fraud and threat detection. These types of relationship visualizations are also useful for patterns and rules discovery.

Use Cases
Semantic Networks



82

Relationships among words and phrases are mapped with graph databases to build semantic networks and discover semantic inferences. These networks, when combined with text mining, are particularly powerful ways to find deep meaning in text data.

Part 6

Summary and Conclusion

Summary of Key Points
References and Resources

Summary of Key Points
A Quick Review

- ✓ Big Data incorporates information that does not originate within the enterprise, and goes deeper than simple business transactions
- ✓ NoSQL brings together relational and non-relational stores under the umbrella of information asset management
- ✓ Traditional BI answers questions about "who," "what," and "when?" Big Data analytics use statistics and data mining to answer questions of "why" and "what if."
- ✓ A data model is a tool for people to find, use and manage information assets. Models are essential for data storage, governance, business requirements, change management, and program scope
- ✓ In traditional data projects we model before storing data; in big data projects we often store data first and then model it.
- ✓ Non-relational data stores may be used to capture data, explore it for business value, and augment or extend the data warehouse.
- ✓ Data models capture information about things, identities, properties, associations and metrics.
- ✓ Key-value stores, document stores, and graph databases can all be modeled using consistent notation to represent the things, identities, properties, associations and metrics contained within them.

References and Resources
To Learn More

Data Modeling for MongoDB
Steve Hoberman, Technics Publications, 2014

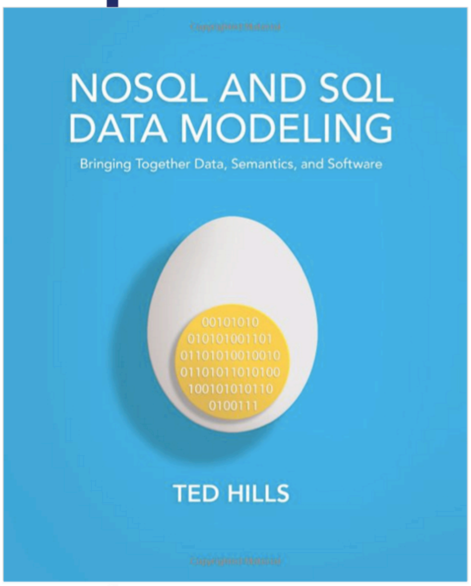
Graph Databases
Ian Robinson, Jim Weber & Emil Eifrem, O'Reilly, 2013

Making Sense of NoSQL: A Guide for Managers and the Rest of Us
Dan McCreary& Ann Kelly, Manning Publications, 2013

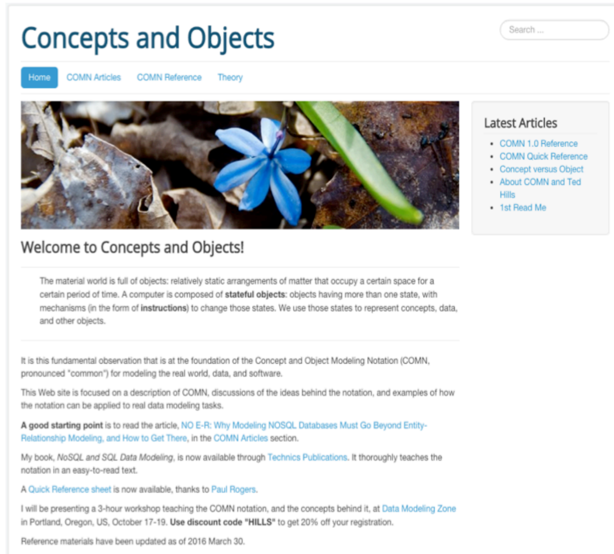
Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement
Eric Redmond & Jim Wilson, Pragmatic Programmers LLC, 2012

References and Resources

To Learn More



tewdur.com



Thank You!

Dave Wells (dwells@infocentric.org)