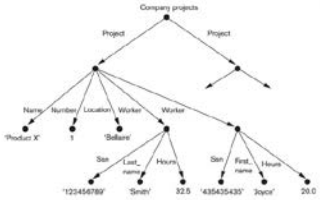


CZ2007
Introduction to Databases

Semi-Structured Data

Cong Gao

Professor
School of Computer Science and Engineering
Nanyang Technological University, Singapore

Schedule after Recess Week

SQL

- Week 8
- Week 9
- Week 10
- Week 11

Semi-Structured Data

- Week 12 (No Lecture on 11 April)
- Week 13

Quiz-2

- Week 12
- Quiz during **Tutorial** session
- Quiz syllabus: The first 4 weeks for Quiz

Roadmap (Semi-Structured Data)

- Semi-structured Data
- XML
- XML DTD
- JSON

The More Data, The Merrier

Power of Data

- the more data the merrier (GB -> TB -> PB)
- data comes from everywhere in all shapes
- value of data often discovered later

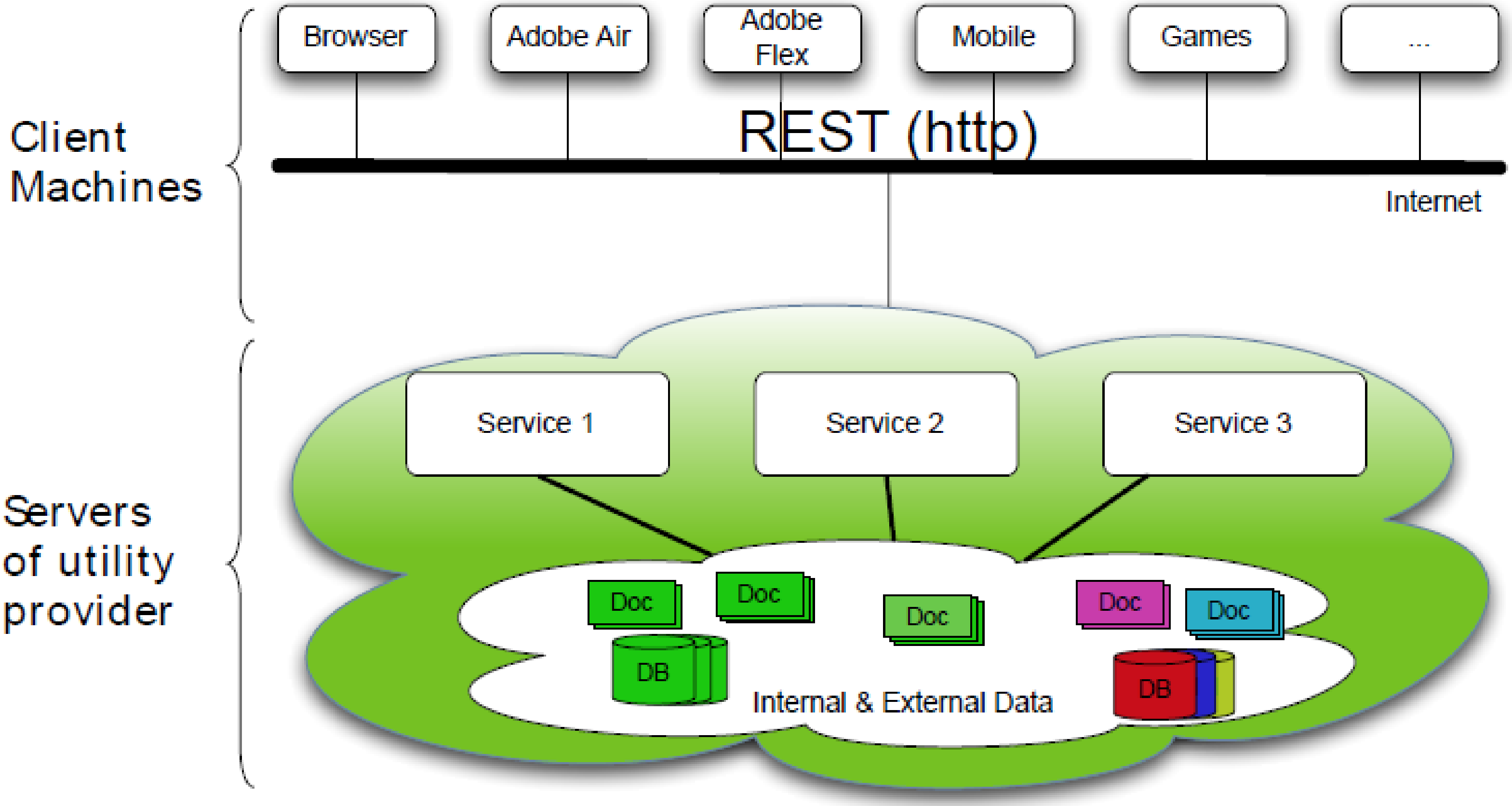
Services turn data into $$$

- the more services the merrier (10 -> 1000 -> 1M -> 1B)
- need to adapt quickly

Goal: Platforms for data and services

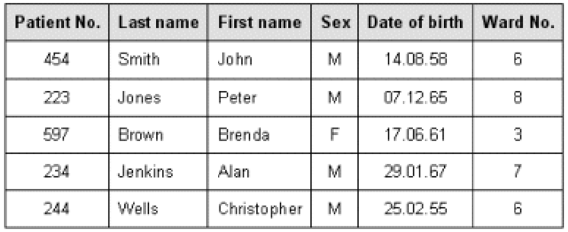
- any data, any service, anywhere and anytime

Data Arrive in Many Shapes



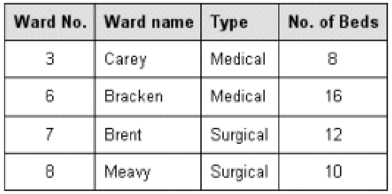
Structured vs. Unstructured Data

Relational databases are highly structured



Patient No.	Last name	First name	Sex	Date of birth	Ward No.
454	Smith	John	M	14.08.58	6
223	Jones	Peter	M	07.12.65	8
597	Brown	Brenda	F	17.06.61	3
234	Jenkins	Alan	M	29.01.67	7
244	Wells	Christopher	M	25.02.55	6

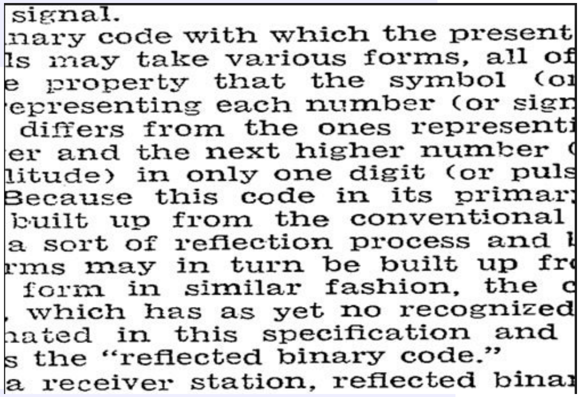
- All data resides in tables
- Must define schema before entering data
- Every row confirms to the table schema
- Changing the schema is hard and may break many things



Ward No.	Ward name	Type	No. of Beds
3	Carey	Medical	8
6	Bracken	Medical	16
7	Brent	Surgical	12
8	Meavy	Surgical	10

Texts are highly unstructured

- Data is free-form
- No schema and it’s hard to define one
- Readers need to infer structures & meanings



signal.
nary code with which the present
ls may take various forms, all of
e property that the symbol (or
epresenting each number (or sign
differs from the ones representi
er and the next higher number (
litude) in only one digit (or puls
Because this code in its primar
built up from the conventional
a sort of reflection process and
rms may in turn be built up fr
form in similar fashion, the c
, which has as yet no recognized
nated in this specification and
s the “reflected binary code.”
a receiver station, reflected binar

What’s in between these two extremes?

Semi-Structured Data

Observation: most data have “some” structure, e.g.

- Book: chapters, sections, titles, paragraphs, references, index, etc.
- Item for sale: name, picture, price, ratings, promotion, etc.
- Web page: HTML

Ideas

- Ensure data is “well-formatted”
- If needed, ensure data is also “well-structured”
  - But make it easy to define and extend this structure
- Make data “self-describing”

A Little Bit of History …

Database world

- 1970 relational databases
- 1990 nested relational model and object oriented databases
- 1995 semi-structured databases

Documents world

- 1974 **SGML** (Structured Generalized Markup Language)
- 1990 **HTML** (Hypertext Markup Language)
- 1992 **URL** (Universal Resource Locator)

Data + documents = information

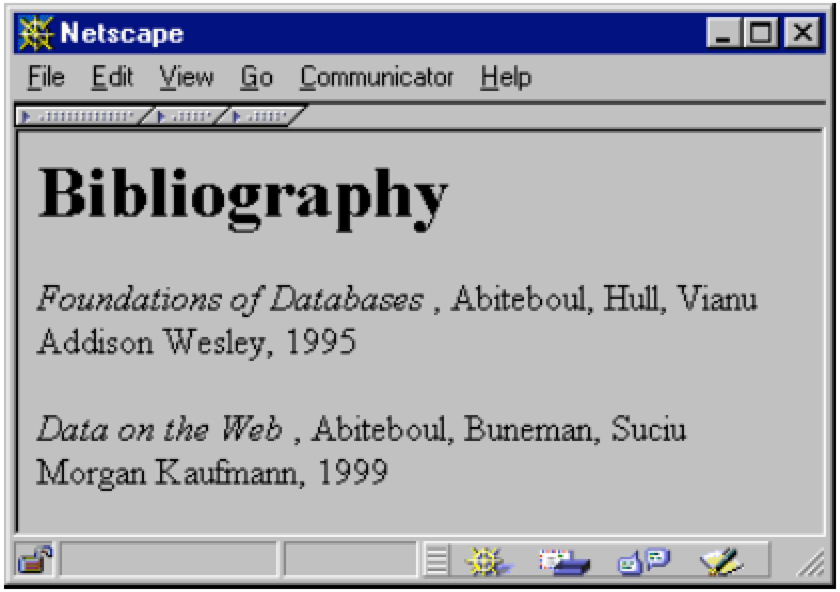
1996 **XML** (Extended Markup Language)

URI (Universal Resource Identifier)

XML as Semi-Structured Data

- XML - The EXtensible Markup Language
- A flexible syntax for data: semi-structured data
- Used in:
  - Configuration files, e.g. Web.Config
  - Replacement for binary formats (MS Word)
  - Document markup: e.g. XHTML
  - Data: data exchange, semistructured data (sensor data, logs, blogs)
- Warning: not normal form! Not even 1NF
- XML is about half as popular as SQL

From HTML to XML



Bibliography

Foundations of Databases , Abiteboul, Hull, Vianu
Addison Wesley, 1995

Data on the Web , Abiteboul, Buneman, Suciu
Morgan Kaufmann, 1999

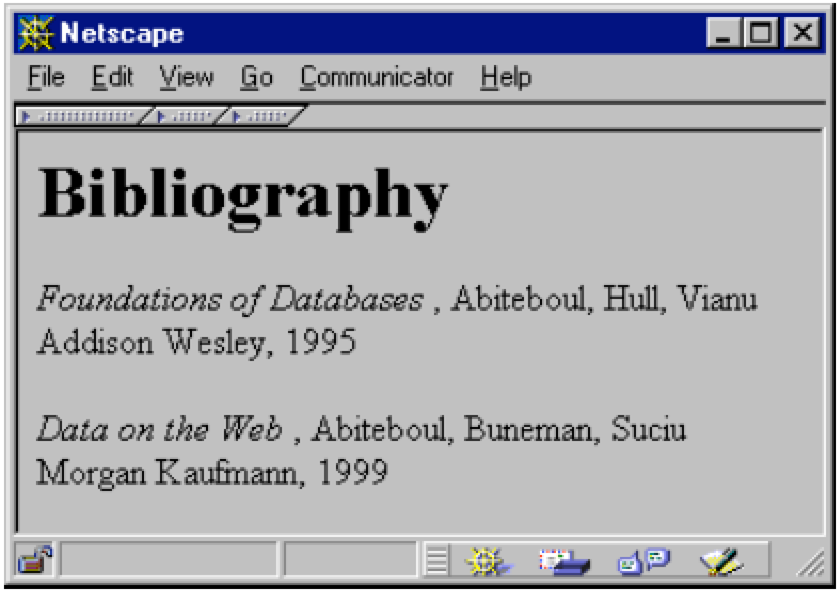
HTML
- The **H**yper**T**ext **M**arkup **L**anguage

HTML describes the presentation

HTML

```
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
      Abiteboul, Hull, Vianu
      <br> Addison Wesley, 1995
<p> <i> Data on the Web </i>
      Abiteoul, Buneman, Suciu
      <br> Morgan Kaufmann, 1999
```

From HTML to XML



Bibliography

Foundations of Databases , Abiteboul, Hull, Vianu
Addison Wesley, 1995

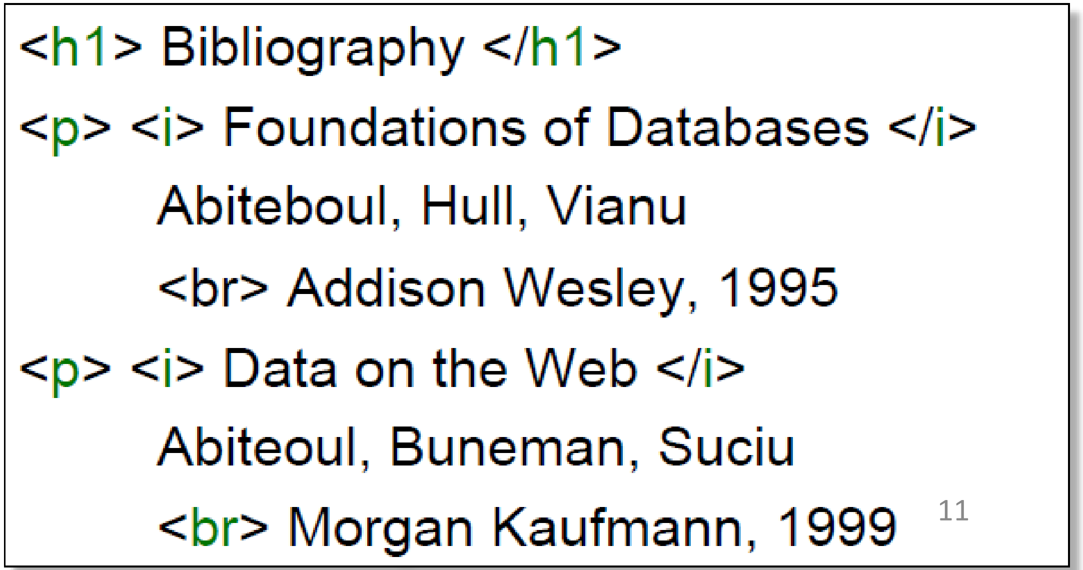
Data on the Web , Abiteboul, Buneman, Suciu
Morgan Kaufmann, 1999

HTML
- The **H**yper**T**ext **M**arkup **L**anguage

HTML describes the presentation

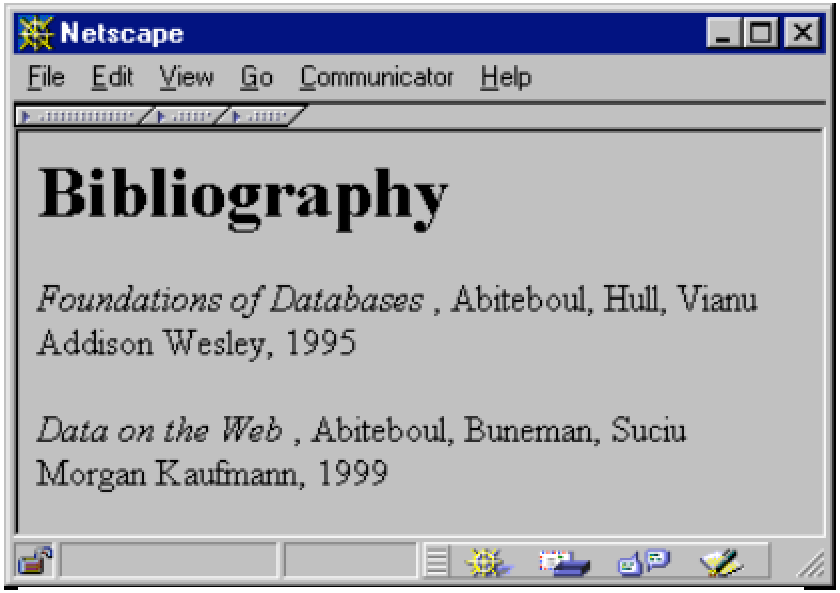
HTML

- It's mostly a "formatting" language
- It mixes presentation and content



```
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
      Abiteboul, Hull, Vianu
      <br> Addison Wesley, 1995
<p> <i> Data on the Web </i>
      Abiteoul, Buneman, Suciu
      <br> Morgan Kaufmann, 1999
```

From HTML to XML



Bibliography

Foundations of Databases , Abiteboul, Hull, Vianu
Addison Wesley, 1995

Data on the Web , Abiteboul, Buneman, Suciu
Morgan Kaufmann, 1999

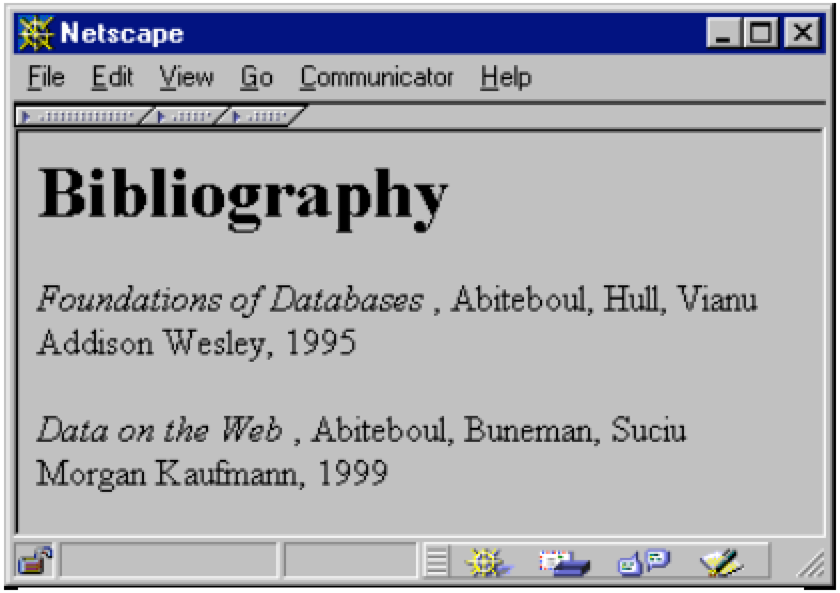
XML describes the content

XML
- The EXtensible Markup Language

XML Syntax

```
<bibliography>
      <book>    <title> Foundations… </title>
                <author> Abiteboul </author>
                <author> Hull </author>
                <author> Vianu </author>
                <publisher> Addison Wesley </publisher>
                <year> 1995 </year>
      </book>
      …
</bibliography>
```

From HTML to XML



Bibliography

Foundations of Databases , Abiteboul, Hull, Vianu
Addison Wesley, 1995

Data on the Web , Abiteboul, Buneman, Suciu
Morgan Kaufmann, 1999

XML
- The EXtensible Markup Language

XML describes the content

XML Syntax

- Text-based
- Capture data (content), not presentation
- Data self-describes its structure
  - Names and nesting of tags have meanings!

```
<bibliography>
    <book>    <title> Foundations… </title>
              <author> Abiteboul </author>
              <author> Hull </author>
              <author> Vianu </author>
              <publisher> Addison Wesley </publisher>
              <year> 1995 </year>
    </book>
    …
</bibliography>
```

HTML vs. XML

Difficulties with HTML ?

- Fixed set of tags
- Elements have document structuring semantics
- For presentation to human readers
- Applications cannot consume and process HTML easily

These difficulties are not in XML

XML Terminology

- Tag names: book, title, …
- Start tags: <book>, <title>, …
- End tags: </book>, </title>, …

- An element is enclosed by a pair of start and end tags: <book>…</book>
- Elements can be nested: <book>…<title>…</title>…</book>
- Empty elements: <is_textbook></is_textbook>

- Can be abbreviated: <is_textbook/>

- Elements can also have attributes: <book ISBN="…" price="80.00">

```
<bibliography>
  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
  </book>…
</bibliography>
```

Ordering generally matters, except for attributes

Well-formed XML documents

A well-formed XML document

- Follows XML lexical conventions

- Wrong: <section>We show that x < 0…</section>

- Right: <section>We show that x &lt; 0…</section>

- Other special entities: > becomes &gt; and & becomes &amp;

- Contains a single root element

- Has properly matched tags and properly nested elements

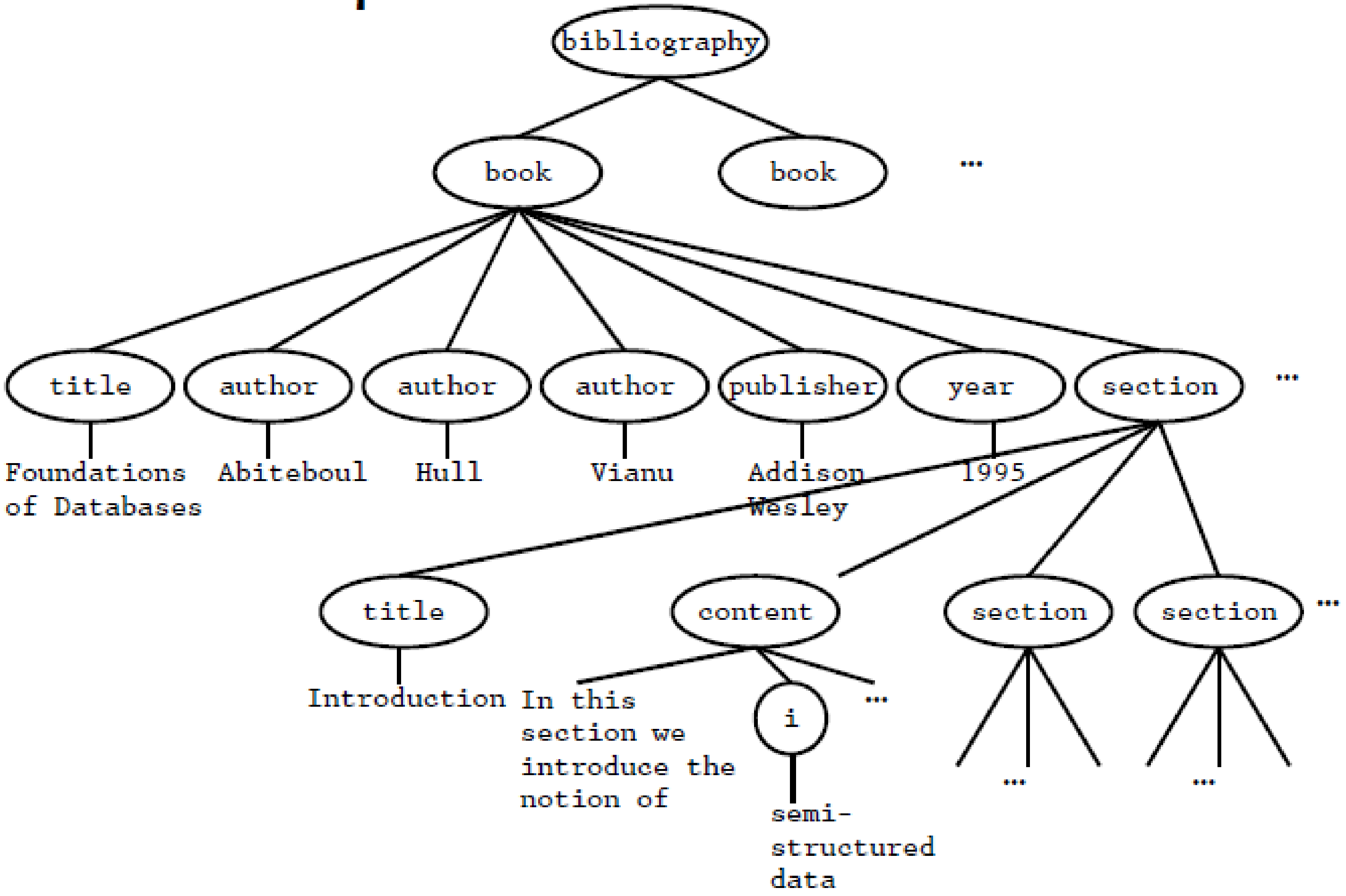
- Right: <section>…<subsection>…</subsection>…</section>

- Wrong: <section>…<subsection>…</section>…</subsection>

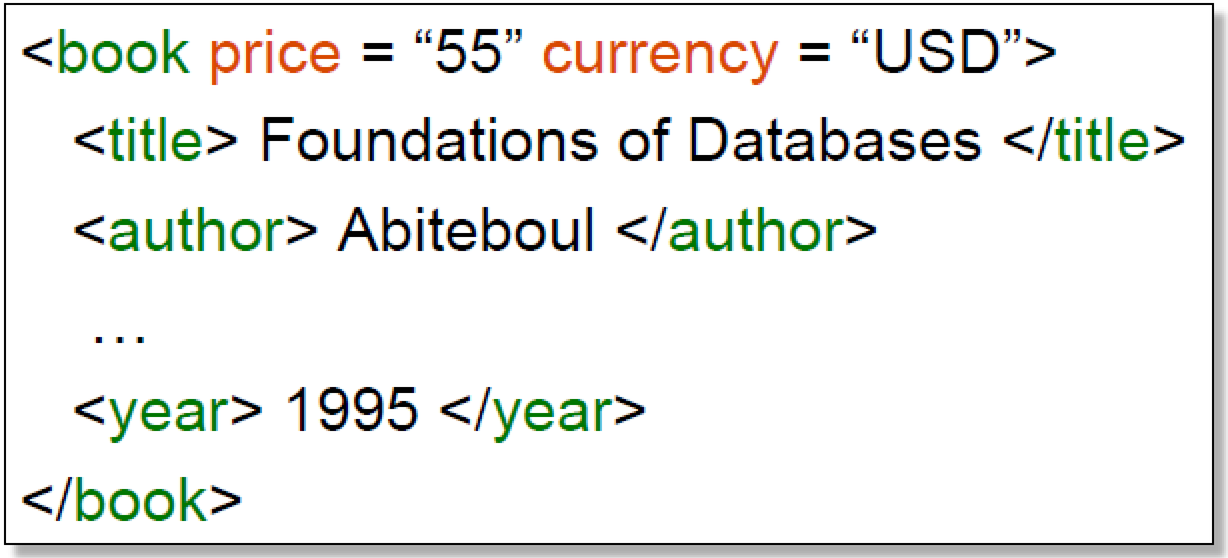
Tree Representation of XML Documents

11

A tree representation



More XML Example: Attributes



```
<book price = “55” currency = “USD”>
   <title> Foundations of Databases </title>
   <author> Abiteboul </author>
    …
   <year> 1995 </year>
</book>
```

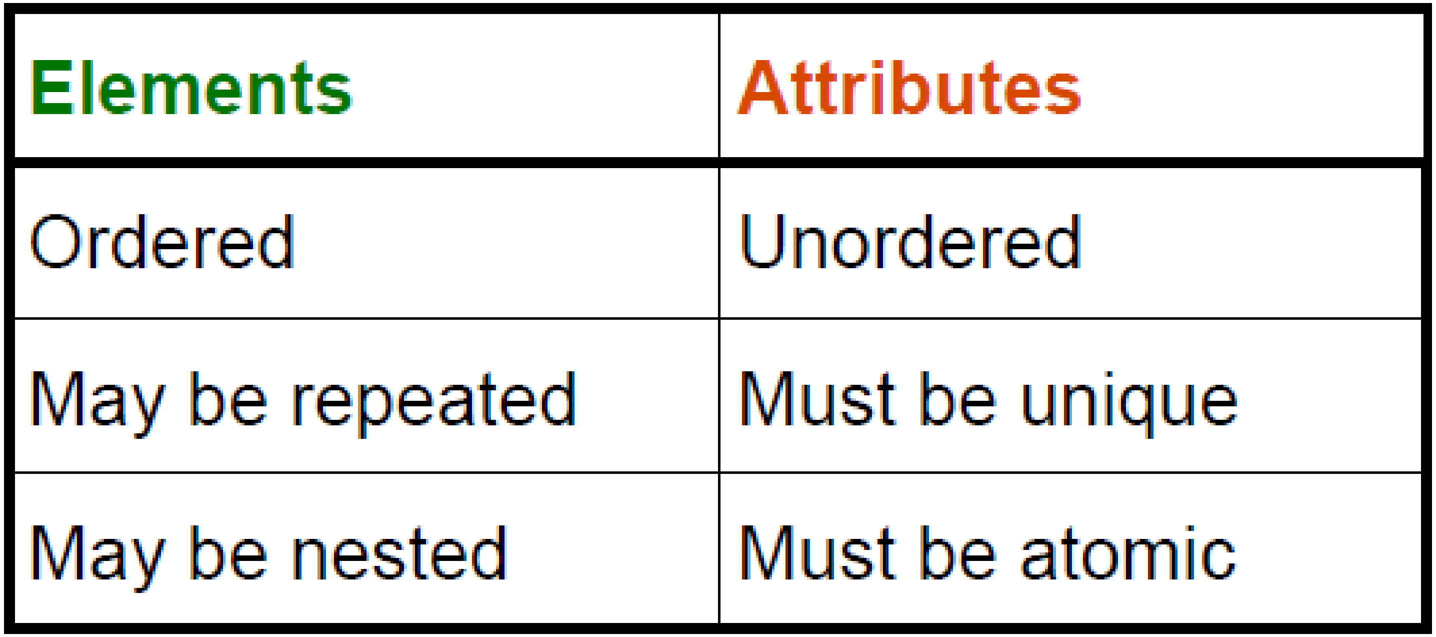
Attributes vs. Elements

```
<book price = “55” currency = “USD”>
   <title> Foundations of DBs </title>
   <author> Abiteboul </author>
    …
   <year> 1995 </year>
</book>
```

```
<book>
  <title> Foundations of DBs </title>
   <author> Abiteboul </author>
    …
   <year> 1995 </year>
   <price> 55 </price>
   <currency> USD </currency>
</book>
```

attributes are alternative ways to represent data

Attributes vs. Elements



Elements	Attributes
Ordered	Unordered
May be repeated	Must be unique
May be nested	Must be atomic

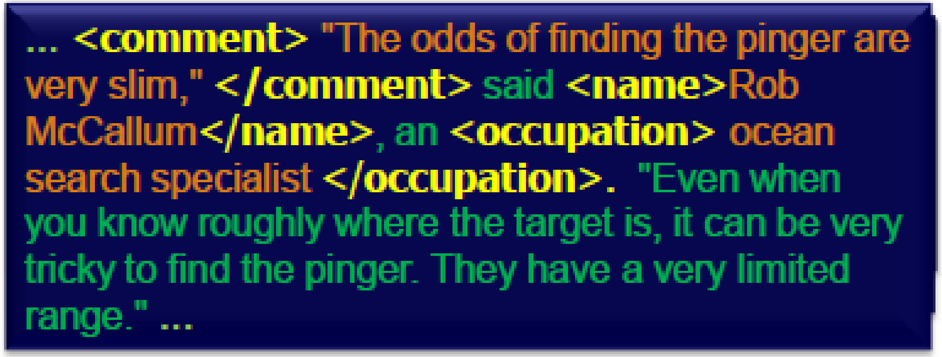
Attribute names must be unique! (No Multisets)

<**person** name = "Wilde" name = "Wutz"/> is illegal!

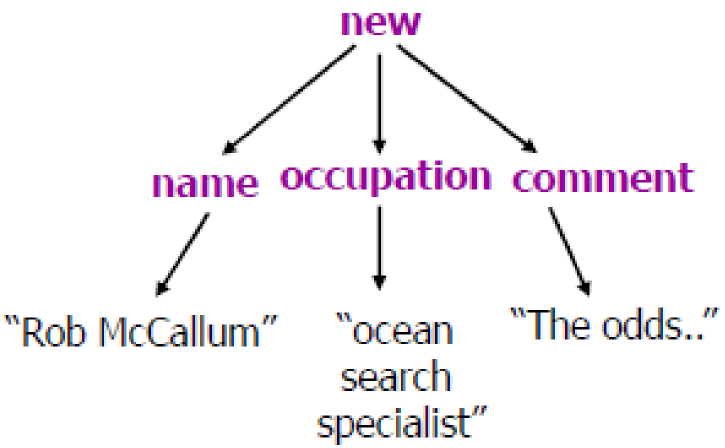
Documents to XML

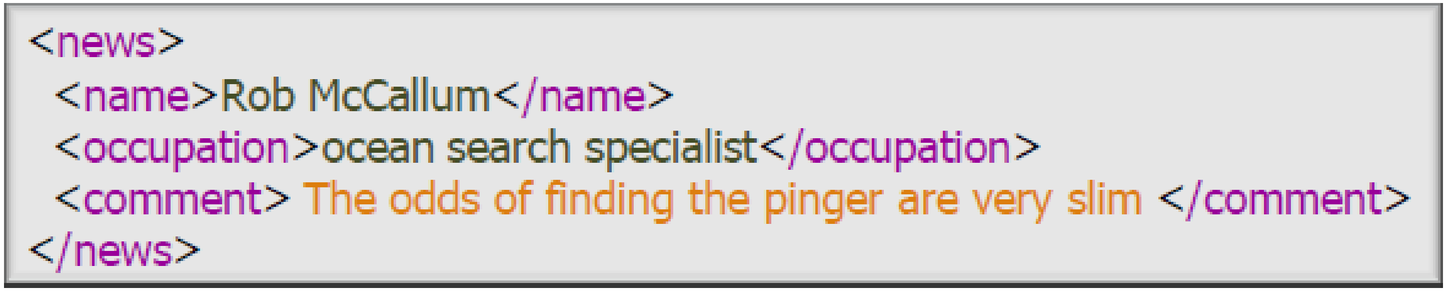
Documents are a quite natural way to represent "objects"

- A great deal of text and semi-structured info



... <comment> "The odds of finding the pinger are very slim," </comment> said <name>Rob McCallum</name>, an <occupation> ocean search specialist </occupation>. "Even when you know roughly where the target is, it can be very tricky to find the pinger. They have a very limited range." ...





```
<news>
  <name>Rob McCallum</name>
  <occupation>ocean search specialist</occupation>
  <comment> The odds of finding the pinger are very slim </comment>
</news>
```

Benefits of XML over Relational Data

- Portability: Just like HTML, you can ship XML data across platforms
- Relational data requires heavy-weight API's

- Flexibility: You can represent any information (structured, semi-structured, documents, …)
- Relational data is best suited for structured data

- Extensibility: Since data describes itself, you can change the schema easily
- Relational schema is rigid and difficult to change

XML vs. Relational Data

- Relational data
  - *Killer application*: Banking
  - Invented as a mathematically clean *abstract data model*
  - *Philosophy*: schema first, then data
- XML
  - First *killer application*: publishing industry
  - Invented as a *syntax for data*, only later an abstract data model
  - *Philosophy*: data and schemas should not be correlated, data can exist with or without schema, or with multiple schemas

XML vs. Relational Data

- Relational data
  - Never had a *standard syntax* for data
  - Strict rules for data *normalization*, flat tables
  - *Order* is irrelevant, textual data supported but not primary goal
- XML
  - *Standard syntax* existed before the data model
  - No data *normalization*, flexibility is a must, nesting is good
  - *Order may* be very important, textual data support a primary goal

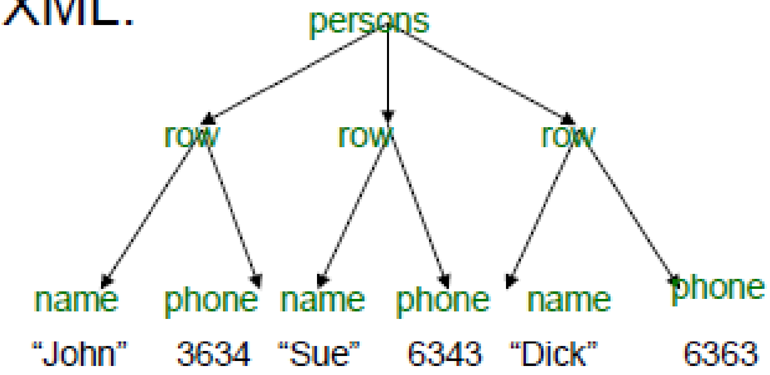
Mapping Relational Data to XML

XML view of relational data

Persons

Name	Phone
John	3634
Sue	6343
Dick	6363

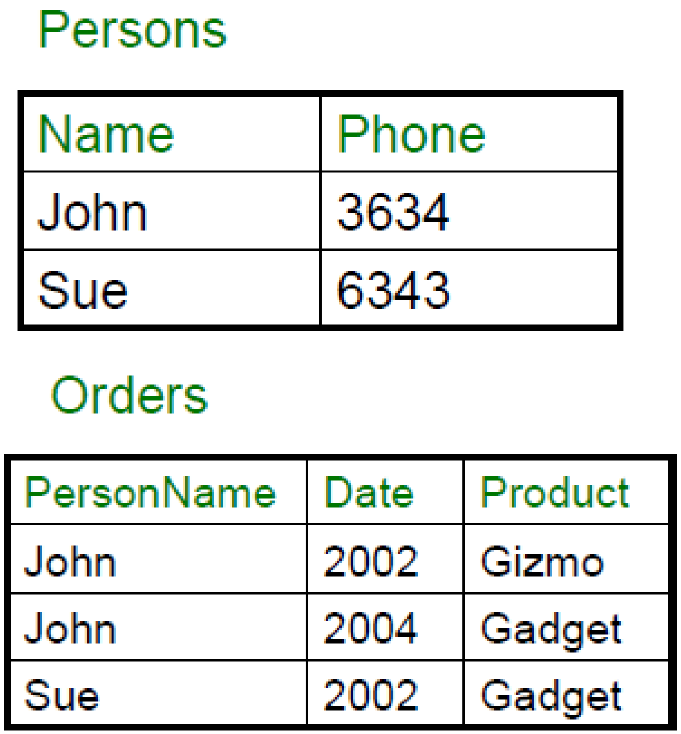
XML:



```
<persons>
     <row> <name>John</name>
                <phone> 3634</phone></row>
      <row> <name>Sue</name>
                 <phone> 6343</phone>
      <row> <name>Dick</name>
                 <phone> 6363</phone></row>
</persons>
```

Mapping Relational Data to XML

XML view of relational data



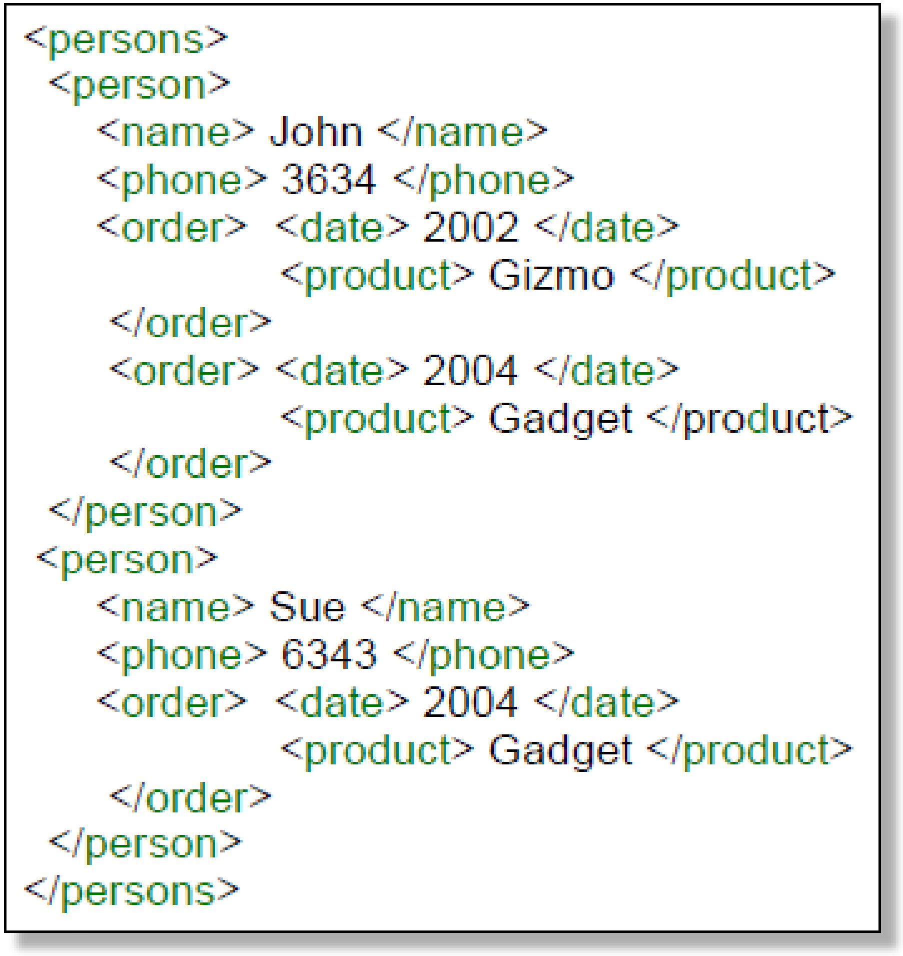
Persons

Name	Phone
John	3634
Sue	6343

Orders

PersonName	Date	Product
John	2002	Gizmo
John	2004	Gadget
Sue	2002	Gadget

XML



```
<persons>
  <person>
     <name> John </name>
     <phone> 3634 </phone>
     <order>  <date> 2002 </date>
              <product> Gizmo </product>
      </order>
      <order> <date> 2004 </date>
              <product> Gadget </product>
      </order>
  </person>
 <person>
     <name> Sue </name>
     <phone> 6343 </phone>
     <order>  <date> 2004 </date>
              <product> Gadget </product>
      </order>
  </person>
</persons>
```

XML is Semi-Structured

- Missing attributes:

```
<person>   <name> John</name>
           <phone>1234</phone>
 </person>

<person>  <name>Joe</name>
</person>
```

no phone !

- Could represent in a table with nulls

name	phone
John	1234
Joe	-

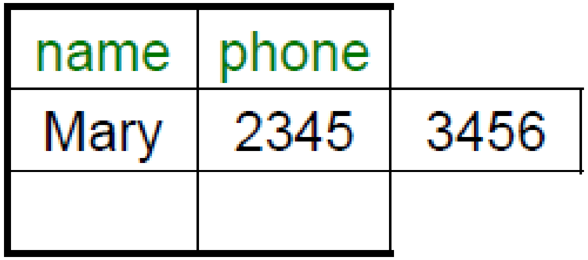
XML is Semi-Structured

- Repeated attributes

```
<person> <name> Mary</name>
         <phone>2345</phone>
         <phone>3456</phone>
</person>
```

Two phones !

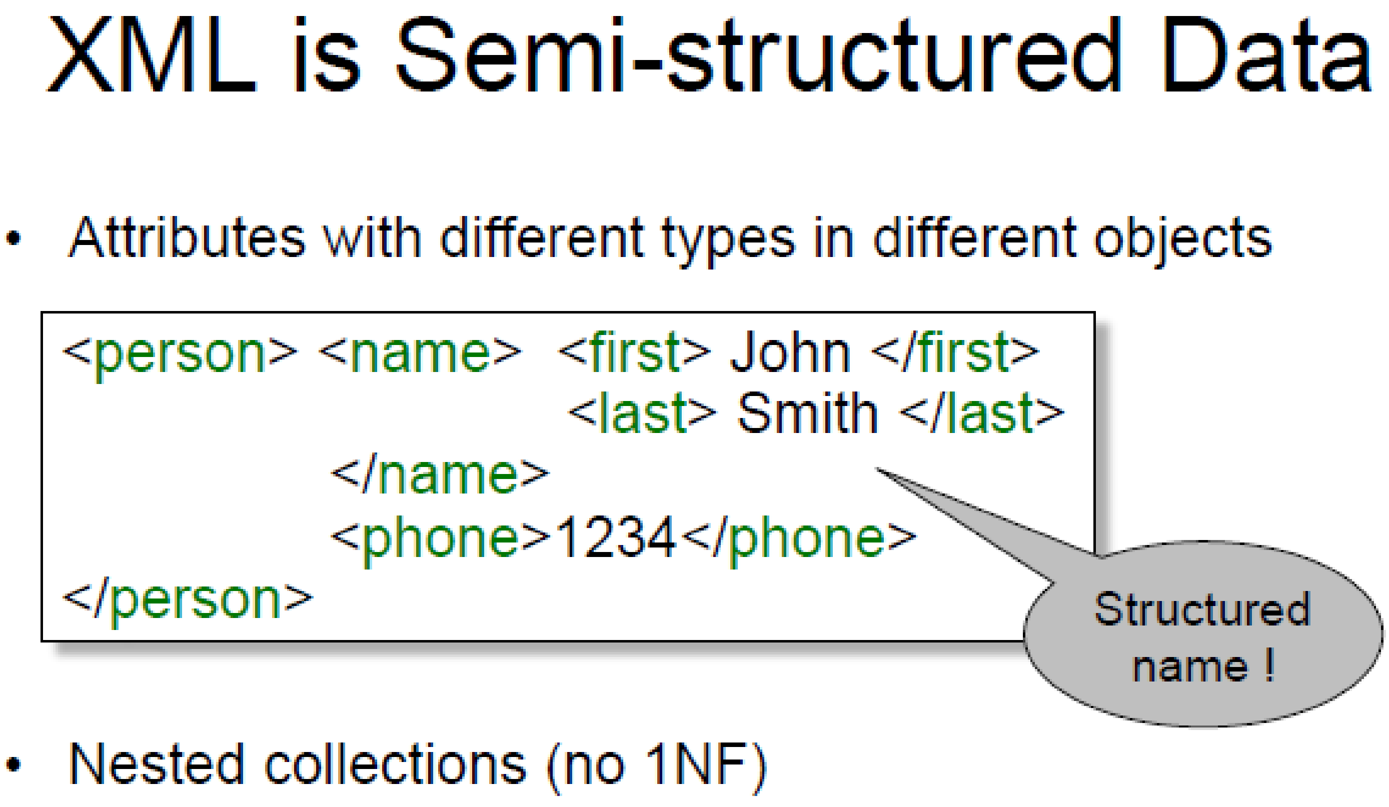
- Impossible in tables:



name	phone	
Mary	2345	3456

???

XML is Semi-Structured



XML is Semi-structured Data

- Attributes with different types in different objects

```
<person> <name>  <first> John </first>
                  <last> Smith </last>
         </name>
         <phone>1234</phone>
</person>
```

Structured name !

- Nested collections (no 1NF)

Questions?

- Semi-structured Data ✓
- XML ✓
- **XML DTD**
- JSON

XML Format Descriptions

- Easy to start with, use your own tags

  - Contrast to relational DB, OO languages

- Only restriction: XML needs to be well-formed

- At some point, this is too much freedom

  - Use same syntax for different documents
  - Facilitate the writing of applications that process data
  - Exchange data with other parties

- Need to restrict the amount of freedom

  - Document Description Methods

Overview of XML Schema Languages

- Several standard Schema Languages
  - **DTDs**, XML Schema, RelaxNG, Schematron

- Schema languages have been designed after, and in an orthogonal fashion, to XML itself

- Schemas and data are decoupled in XML
  - Data can exist with or without schemas
  - Or with multiple schemas
  - Schema evolutions rarely impose evolving the data
  - Schemas can be designed before the data, or extracted from the data

- Makes XML the right choice for manipulating semi-structured data, or rapidly evolving data, or highly customizable data

Document Type Definition (DTD)

Goals:

- Define what tags and attributes are allowed
- Define how they are nested
- Define how they are ordered

Superseded by XML Schema

- Very complex: DTDs still used widely

Element Type Declaration

- Element Types are composed of:
  - Subelements (identified by Name)
  - Attribute lists (identified by Name)
  - Selection of Subelemente (choice)
  - PCDATA **text that WILL be parsed by a parser**
- Quantifier for Subelements and Choice
  - "+" for at least 1
  - "*" for 0 or more
  - "?" for 0 or 1
  - Default: exactly 1
- EMPTY and ANY are special predefined Types

- "|": Declaring either/or Content

<!ELEMENT note (to,from,header,(message|body))>

<!ELEMENT element-name category>

or

<!ELEMENT element-name (element-content)>

Example:
<!ELEMENT br EMPTY>

XML example:
<br />

Element Type Declaration

- Structure: <!ELEMENT *name content*>
- **Example**

```
<!ELEMENT book (title, (author+ | editor), publisher?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author EMPTY>
<!ELEMENT publisher ANY>
```

- Valid document according to this DTD

```
<book >
    <title>Die wilde Wutz</title>
    <author/>  <author></author>
 <publisher><anything>...</anything></publisher>
</book>
```

Declaring Attributes

- An attribute declaration has the following syntax:
- <!ATTLIST element-name attribute-name attribute-type attribute-value>

  DTD example:

  <!ATTLIST payment type CDATA "check">

  XML example:

  <payment type="check" />

attribute-type

The **attribute-type** can be one of the following:

Type	Description
CDATA	The value is character data
(*en1*|*en2*|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

attribute-value

The **attribute-value** can be one of the following:

Value	Explanation
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is optional
#FIXED *value*	The attribute value is fixed

Default Attribute Value

DTD:

```
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
```

Valid XML:

```
<square width="100" />
```

In the example above, the "square" element is defined to be an empty element with a "width" attribute of type CDATA. If no width is specified, it has a default value of 0.

Attribute type--#REQUIRED

- Syntax: <!ATTLIST element-name attribute-name attribute-type #REQUIRED>
- Example
- DTD:
  <!ATTLIST person number CDATA #REQUIRED>

  Valid XML:
  <person number="5677" />

  Invalid XML:
  <person />
- Use the #REQUIRED keyword if you don't have an option for a default value, but still want to force the attribute to be present.

Attribute type-- #IMPLIED

- Syntax: <!ATTLIST element-name attribute-name attribute-type #IMPLIED>
- Example
- DTD:
  <!ATTLIST contact fax CDATA #IMPLIED>

  Valid XML:
  <contact fax="555-667788" />

  Valid XML:
  <contact />
- Use the #IMPLIED keyword if you don't want to force the author to include an attribute, and you don't have an option for a default value.

Attribute type (#fixed)

- Syntax
- <!ATTLIST element-name attribute-name attribute-type #FIXED "value">
- Example
- DTD:
  <!ATTLIST sender company CDATA #FIXED "Microsoft">

  Valid XML:
  <sender company="Microsoft" />

  Invalid XML:
  <sender company="W3Schools" />
- Use the #FIXED keyword when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.

Attribute Lists

- Structure: <!ATTLIST *ElementName definition*>

-
  ```
  <!ATTLIST book
          isbn    ID          #REQUIRED
          price   CDATA   #IMPLIED
          curr    CDATA   #FIXED "EUR"
          index  IDREFS   "" >
  ```
- Valid and Not-valid Books

  ```
  <book isbn="abc" curr="EUR"/>  !! no price
  <book isbn="abc" price="30"/>  !! Curr, index default
  <book index="DE" isbn="abc" curr="EUR"/>
  <book/>   !! Missing isbn Attribute
  <book isbn="abc" curr="USD"/>  !! wrong currency
  ```

Entity

Entity References	Character
&lt;	<
&gt;	>
&amp;	&
&quot;	"
&apos;	'

- Syntax: <!ENTITY entity-name "entity-value">
- DTD Example:
  <!ENTITY writer "Donald Duck.">
  <!ENTITY copyright "Copyright W3Schools.">
- XML example:
  <author>&writer;&copyright;</author>

Note: An entity has three parts: an ampersand (&), an entity name, and a semicolon (;).

Entity

- An External Entity Declaration
- Syntax <!ENTITY entity-name SYSTEM "URI/URL">
- Example
- DTD Example:

```
<!ENTITY writer SYSTEM
"https://www.w3schools.com/entities.dtd">
<!ENTITY copyright SYSTEM
"https://www.w3schools.com/entities.dtd">
```

- XML example:

```
<author>&writer;&copyright;</author>
```

DTD Example

```
<!ELEMENT book (title, (author+ | editor), publisher?)>
<!ATTLIST book
   year CDATA #REQUIRED
   isbn   ID           #REQUIRED
   price   CDATA   #IMPLIED
   curr    CDATA   #FIXED "EUR"
   index  IDREFS    "" >
<!ELEMENT author (firstname, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

SUMMARY

- Semi-structured Data ✓
- XML ✓
- XML DTD ✓
- **JSON**

Difficulties with XML

- “Tree, and not a graph.”
  - Difficulty in modeling N:M relationships
  - The notion of reference (e.g. XLink, XPointer) not well integrated in the XML stack
- “Duplication of concepts”
  - Many ways to do the same thing
  - Justification for a “simpler” data model like RDF
- “Concepts that *seem* logically unnecessary”
  - PIs, comments, documents, etc
- Additional complexity factors
  - xsi:nil, QName in content, etc
- “Boring”
  - so is the (enterprise) world where XML lives

Other Semi-Structured Data

- JSON
- CSV
- Avro
- Protocol Buffers
- RDF
- Property Graphs
- …

Why do we still talk about XML ?

- It is a standard (not owned by anybody)
- Very well documented
- Many tools available
- Mother of all semi-structured data
- has the most features
- XML is here to stay
- It actually works!

JSON

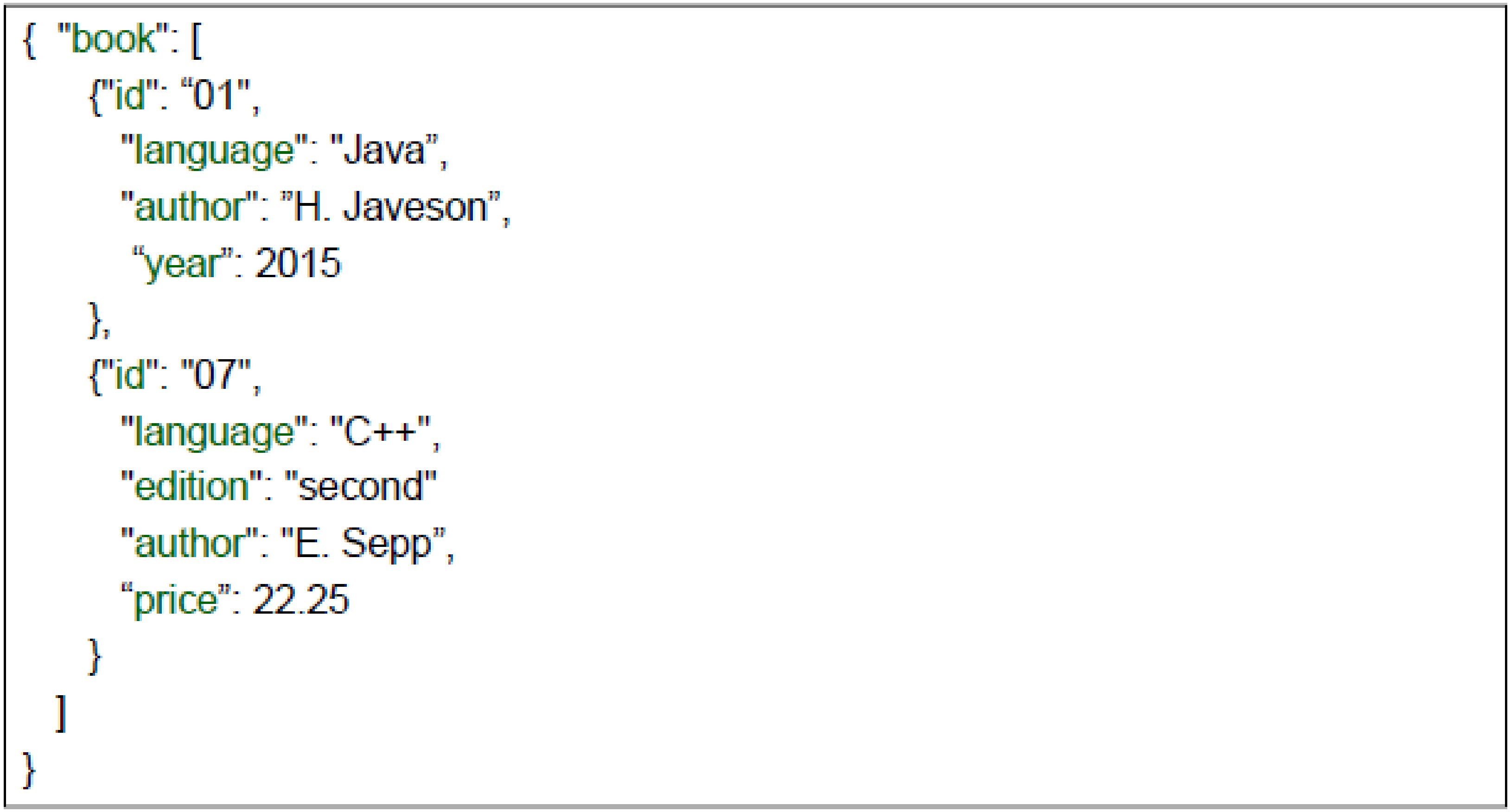
JSON

- JavaScript Object Notation
  - lightweight text-based open standard designed for human-readable data interchange.
- Interfaces in C, C++, Java, Python, Perl, etc.
- The filename extension is .json.

Semistructured data model

- Flexible, nested structure (trees)
- Does not require predefined schema ("self describing”)
- Text representation: good for exchange, bad for performance
- Most common use: Language API

JSON - Syntax



```
{  "book": [
     {"id": “01",
        "language": "Java”,
        "author": ”H. Javeson”,
         “year”: 2015
     },
     {"id": "07",
        "language": "C++",
        "edition": "second"
        "author": "E. Sepp”,
        “price”: 22.25
     }
   ]
}
```

JSON - Terminology

Curly braces

- Hold objects
- Each object is a list of name/value pairs separated
- by , (comma)
- Each pair is a name is followed by ':’ (colon) followed by the value

Square brackets

- Hold arrays and values are separated by , (comma).

What is the data made up of?

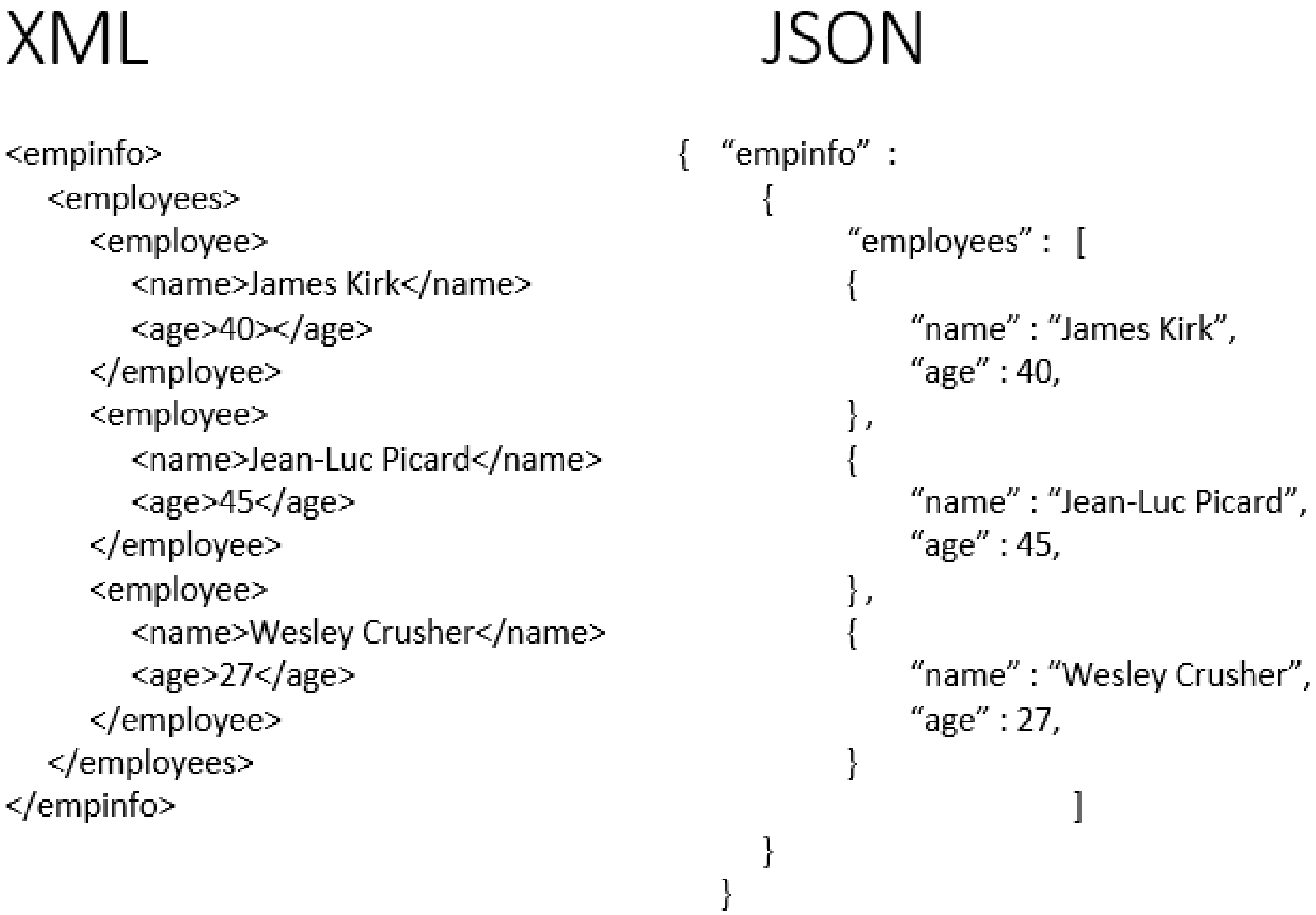
- Objects, lists, and atomic values (integers, floats, strings, booleans).

JSON – Data Structure

Collection

- Collections of name-value pairs:
  - {“name1”: value1, “name2”: value2, …}
- The “name” is also called a “key”
- Ordered lists of values: [obj1, obj2, obj3, ...]

XML vs. JSON



XML

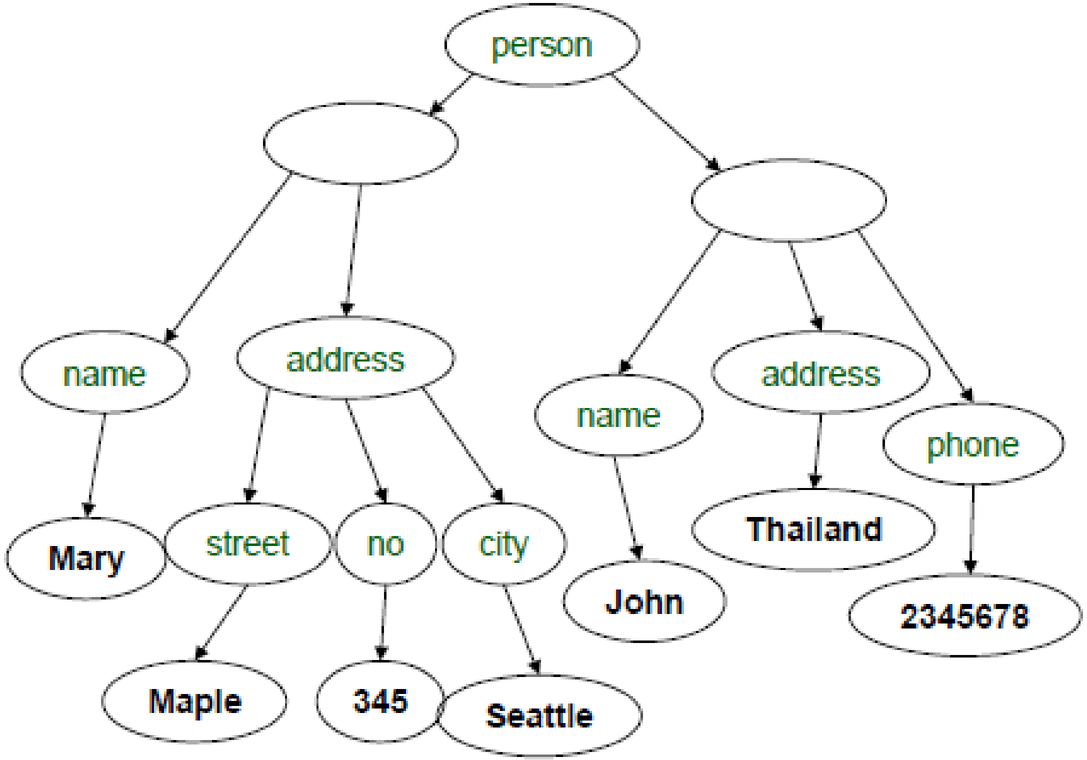
```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{   "empinfo" :
        {
              "employees" :  [
              {
                   "name" : "James Kirk",
                   "age" : 40,
              } ,
              {
                   "name" : "Jean-Luc Picard",
                   "age" : 45,
              } ,
              {
                   "name" : "Wesley Crusher",
                   "age" : 27,
              }
                                  ]
        }
   }
```

Tree View of JSON Data

```
{“person”:
  [ {“name”: “Mary”,
     “address”:
        {“street”: “Maple”,
          “no”: 345,
          “city”: “Seattle”}},
    {“name”: “John”,
     “address”: “Thailand”,
     “phone”: 2345678}
    ]
}
```

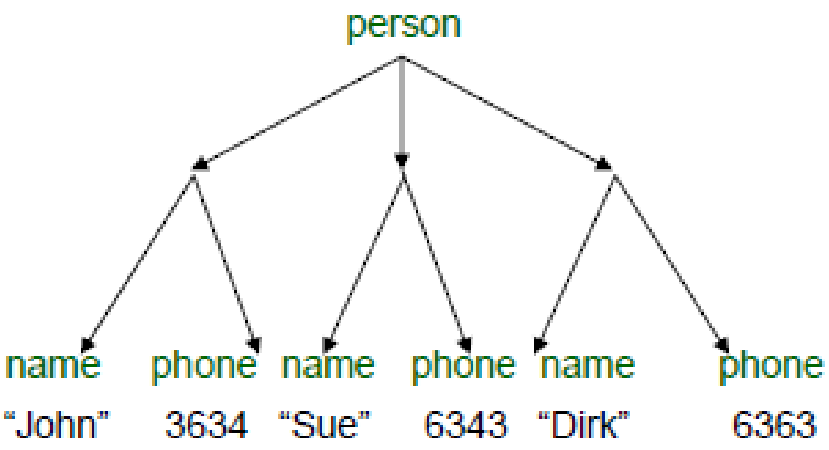


Self-describing

Mapping Relational Data to JSON

Person

name	phone
John	3634
Sue	6343
Dirk	6363



```
{“person”:
    [{“name”: “John”, “phone”: 3634},
     {“name”: “Sue”,  “phone”: 6343},
     {“name”: “Dirk”,  “phone”: 6383}
    ]
}
```

Mapping Relational Data to JSON

Person

name	phone
John	3634
Sue	6343

Orders

personName	date	product
John	2002	Gizmo
John	2004	Gadget
Sue	2002	Gadget

```
{"Person":
     [{"name": "John",
       "phone": 3646,
       "Orders": [{"date": 2002,
                   "product": "Gizmo"},
                  {"date": 2004,
                   "product": "Gadget"}
                 ]
      },
      {"name": "Sue",
       "phone": 6343,
       "Orders": [{"date": 2002,
                    "product": "Gadget"}
                 ]
      }
     ]
}
```

Handling NULL and Repeated Values

name	phone
John	1234
Joe	-

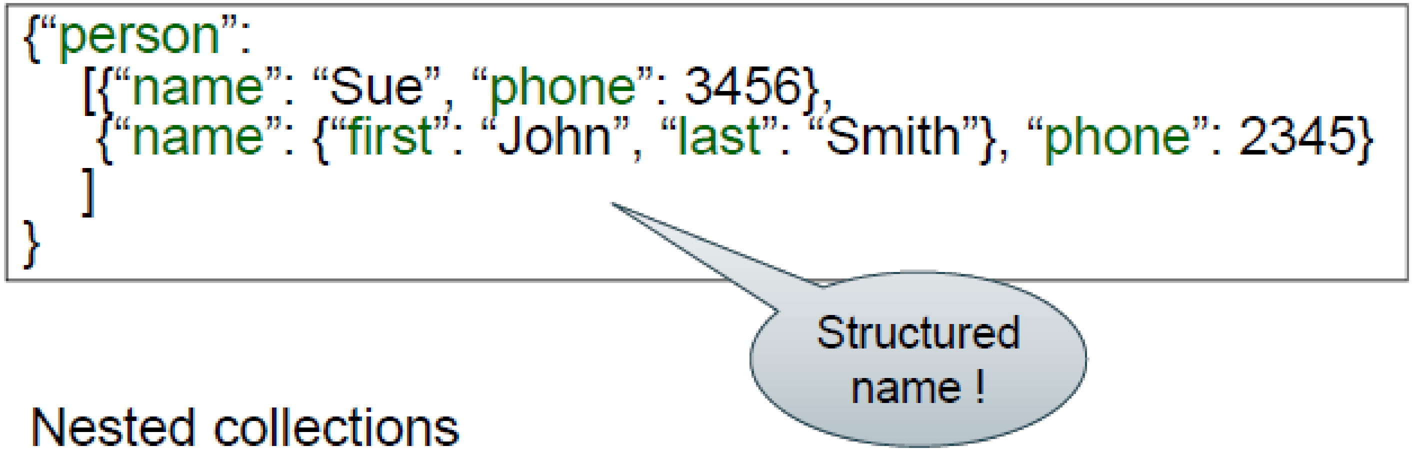
```
{“person”:
    [{“name”: “John”, “phone”: 1234},
     {“name”: “Joe”}]
}
```

no phone !

```
{“person”:
    [{“name”: “John”, “phone”: 1234},
     {“name”: “Mary”, “phone”: [1234, 5678]}]
}
```

Two phones !

Handling Heterogeneous Objects



```
{“person”:
    [{“name”: “Sue”, “phone”: 3456},
     {“name”: {“first”: “John”, “last”: “Smith”}, “phone”: 2345}
    ]
}
```

Structured name !

- Nested collections
- Heterogeneous collections

Summary

Data Exchange Format

- Well suited for exchanging data between applications
- XML, JSON

Data Models

- Some systems use them as data models
- SQL Server – supports XML-valued relations
- CouchBase, Mongodb – JSON as data model

Query Languages

- Xpath, Xquery
- CouchBase – N1QL
- JSONiq

Will NOT discuss in this lecture!

Questions ??

Thank You !