

TDDD43

Theme 1: Semi-structured data, XML and RDF

Fang Wei-Kleiner
<http://www.ida.liu.se/~TDDD43>

Plan for Theme 1

- Data model: semi-structured data, XML, RDF, schema, data guide
- Query languages: XPath, XQuery, XSLT, SPARQL
- XML Query processing techniques

Today's lecture

- Introduction to semi-structured data
- Technologies
 - XML/RDF
- Defining the data model
 - Data model vs. Data guides
- Schema technologies
 - DTD/XML Schema/RDF Schema

Semi-structured data

- Data is not just text, but is not as well-structured as data in databases
- Occurs often in web databases
- Occurs often in integration of databases

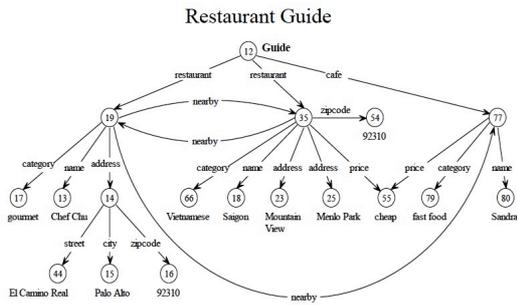
Semi-structured data properties

- Irregular structure
 - Heterogeneous elements, some elements may be incomplete.
 - Missing fields.
- Implicit structure
 - Hidden in the texts.
 - Computation required to obtain the structure (e.g. parsing)
- Partial structure
 - Bitmaps
 - Unstructured text

OEM (Object Exchange Model)

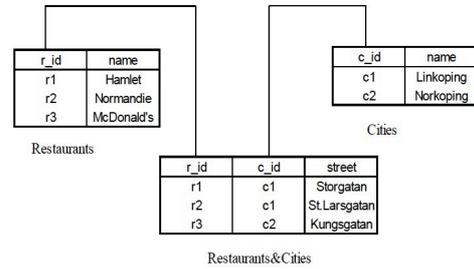
- Graph
- Nodes: objects
 - Oid
 - value: atomic or complex
 - atoms: integer, string, gif, html, ...
 - value of a complex object is a set of object references (label, oid)
- Edges have labels
- OEM is used by a number of systems (ex. Lorel)

OEM example

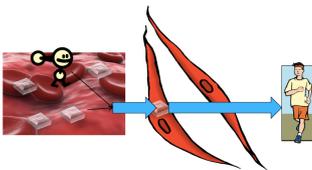


Exercise OEM

Represent the relations below using the OEM data model.



XML motivation example



Muscle cell

Describe the transformation process of sugar in the blood to the energy in muscle cells.

Relational modeling

Compartment	
Id	Name
Blood	Inblood
Cell	Musclecell

Reaction	
Id	Name
Tocell	Sugartocell
Move	Makemovement

Reactant	
Reaction	Species
ToCell	Sug1
ToCell	Ins
Move	Sug2

Species		
Id	Name	Compartment
Sug1	Sugar	Blood
Ins	Insulin	Blood
Sug2	Suga	Cell
En	Energy	Cell

Product	
Reaction	Species
ToCell	Sug2
Move	En

Modeling problem

- Far from semi-structured proposal
 - Not suitable for describing tree structure
 - Too general or many tables
- Static – all attributes typed
- All data entries atomic – in principle

XML representation

- Ordered tree
 - Similar to semi-structured proposal
- Element vs. Attribute
- Extensible
 - New kinds of data can be integrated
- Flexible
 - Easy to mix different kinds of data

```
<?xml version="1.0" encoding="UTF-8"?>
<minimodel name="sugartransport"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns="http://www.w3.org/2001/XMLSchema-instance"
/>
<listOfCompartments>
<compartment id="blood" name="inblood" />
<compartment id="cell" name="musclecell" />
</listOfCompartments>
<listOfSpecies>
<species id="sug1" name="sugar" compartment="blood" />
<species id="ins" name="insulin" compartment="blood" />
<species id="sug2" name="sugar" compartment="cell" />
<species id="en" name="energy" compartment="cell" />
</listOfSpecies>
<listOfReactions>
<reaction id="tocell" name="sugartocell">
<listOfReactants>
<speciesReference species="sug1" />
<speciesReference species="ins" />
</listOfReactants>
<listOfProducts>
<speciesReference species="sug2" />
</listOfProducts>
</reaction>
<reaction id="move" name="makemovement">
<listOfReactants>
<speciesReference species="sug2" />
</listOfReactants>
<listOfProducts>
<speciesReference species="en" />
</listOfProducts>
</reaction>
</listOfReactions>
</minimodel>
```

```

<?xml version="1.0" encoding="UTF-8"?>
<minimodel name="sugartransport"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="minimodel.xsd">
  <listOfCompartments>
    <compartment id="blood" name="inblood" />
    <compartment id="cell" name="musclecell" />
  </listOfCompartments>
  <listOfSpecies>
    <species id="sug1" name="sugar" compartment="blood" />
    <species id="ins" name="insulin" compartment="blood" />
    <species id="sug2" name="sugar" compartment="cell" />
    <species id="en" name="energy" compartment="cell" />
  </listOfSpecies>
  <listOfReactions>
    <reaction id="tocell" name="sugartocell">
      <listOfReactants>
        <speciesReference species="sug1" />
        <speciesReference species="ins" />
      </listOfReactants>
      <listOfProducts>
        <speciesReference species="sug2" />
      </listOfProducts>
    </reaction>
    <reaction id="move" name="makemovement">
      <listOfReactants>
        <speciesReference species="sug2" />
      </listOfReactants>
      <listOfProducts>
        <speciesReference species="en" />
      </listOfProducts>
    </reaction>
  </listOfReactions>
</minimodel>

```

- Tag: reaction
- Start tag: <reaction>
- End tag: </reaction>
- Element: <reaction>...</reaction>
- Elements can be nested
- Attribute: <reaction id= "tocell"
- Processing instructions for apps: <? ...java applet... ?>
- Namespaces allow external schemas and qualified names
XmInS:xsi=" ...

Defining the XML model: DTD

- A Document Type Definition (DTD) defines the legal building blocks of an XML document.
- It defines the document structure with a list of legal elements and attributes.
- In the DTD all XML documents are one of:
 - Elements
 - Attributes
 - Entities
 - PCDATA – parsed character data
 - CDATA – character data

DTD example

```

<!DOCTYPE bibliography [
  <!ELEMENT bibliography (book+)>
  <!ELEMENT book (title, author*, publisher?, year?, section*)>
  <!ATTLIST book ISBN CDATA #REQUIRED>
  <!ATTLIST book price CDATA #IMPLIED>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT author (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT i (#PCDATA)>
  <!ELEMENT content (#PCDATA|i)*>
  <!ELEMENT section (title, content?, section*)>
]
+ : 1+ * : 0+, ? : 0 or 1
#IMPLIED : optional

```

XML Schema

- The XML Schema defines the legal building blocks of an XML document.
- An XML Schema:
 - defines elements
 - defines attributes
 - defines which elements are child elements
 - defines the order of child elements
 - defines the number of child elements
 - defines data types for elements and attributes
 - defines default and fixed values for elements and attributes

XML Schema vs. DTD

- XML Schemas are extensible to future additions
 - extend element definitions
- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML
- XML Schemas support data types
- XML Schemas support namespaces

Why use DTD or XML Schema?

- Benefits of not using them
 - Unstructured data is easy to represent
 - Overhead of validation is avoided
- Benefits of using them
 - Serve as schema for the XML data
 - Guards against errors
 - Helps with processing
 - Facilitate information exchange
 - People can agree to use a common DTD or XML Schema to exchange data (e.g., XHTML)

RDF: Resource Description Framework

- Framework for describing resources on the web
- Designed to be read and understood by computers
- Not designed for being displayed to people
- Written in XML
- RDF is a W3C Recommendation

RDF example

```
<?xml version="1.0" encoding="UTF-8"?>
<species metaid="_506372" id="E1" name="MAPKKK activator"
compartment="compartment"
initialConcentration="3e-05">
  <annotation>
    <rdf.RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:bqbiol="http://biomodels.net/biology-qualifiers/"
      xmlns:bqmodel="http://biomodels.net/model-qualifiers/">
      <rdf.Description rdf:about="#_506372">
        <bqbiol.isVersionOf>
          <rdf.Bag>
            <rdf.li rdf:resource="http://www.ebi.ac.uk/interpro/#IPR003577"/>
          </rdf.Bag>
        </bqbiol.isVersionOf>
      </rdf.Description>
    </rdf.RDF>
  </annotation>
</species>
```

RDF data model: triples

- A **Resource** is anything that can have a URI, such as our molecule "_506372"
- A **Property** is a Resource that has a name, such as "isVersionOf"
- A **Property value** is the value of a Property, such as "IPR003577"
- (note that a property value can be another resource)

Suitable for semi-structured data.

RDF Schema

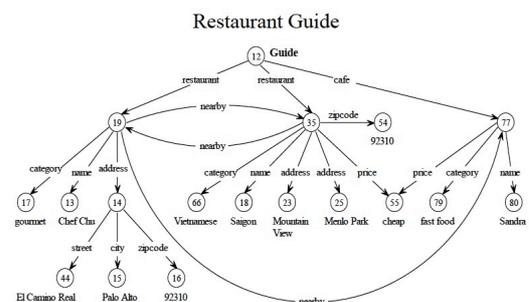
```
<?xml version="1.0" encoding="UTF-8"?>
<rdf.RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <rdf.Description rdf:ID="species">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf.Description>
  <rdf.Description rdf:ID="protein">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#species"/>
  </rdf.Description>
</rdf.RDF>
```

Define relations between objects

Data guides

- A structural summary over a data source that is used as a dynamic schema
- Is used in query formulation and optimization
- Is often created a posteriori
- Properties:
 - concise
 - accurate
 - convenient

OEM example



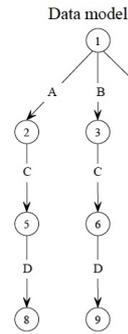
Data guide definition

- **Label path**: sequence of labels L_1, L_2, \dots, L_n from root
- **Data path**: alternating sequence of labels and oids $L_1, o_1, L_2, o_2, \dots, L_n, o_n$ from root
- Data path d is an **instance** of label path l if the sequences of labels are identical in l and d .

A **data guide** for object s is an object d such that

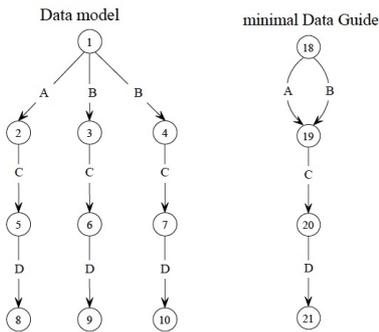
- (1) every label path of s has exact one data path instance in d , and
- (2) each label path in d is a label path of s .

Multiple data guides

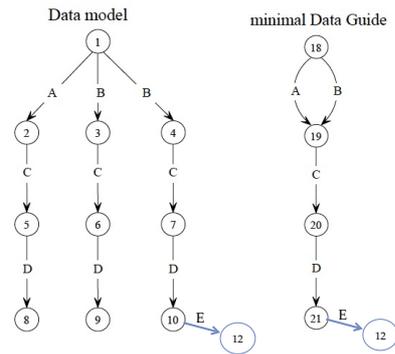


How many data guides are there for this graph instance?

Multiple data guides



Minimal data guide



Update on the data
Update on data guide

Problem: it is not a
Data guide anymore!

Strong data guide

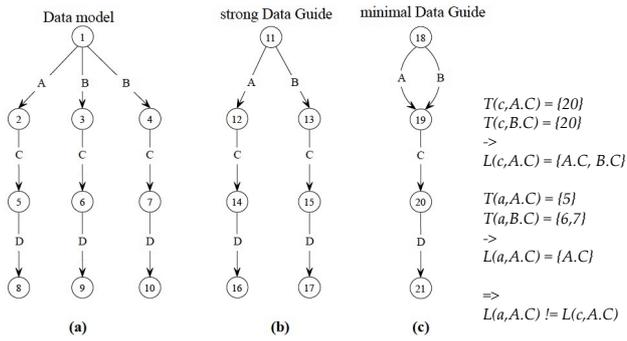
Intuition:

- Two distinct label paths are indistinguishable if they reach the same vertex set in the data graph.
- If two label paths are indistinguishable in the data guide G , they should also be indistinguishable in the original data graph DB .

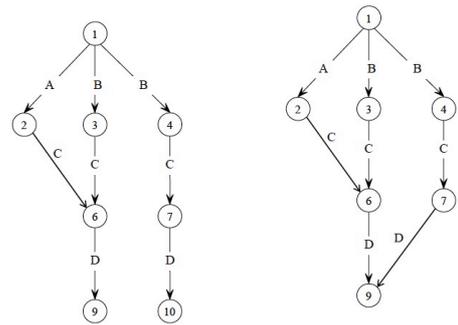
Strong data guide

- An object o can be **reached** from s via l if there is a data path of s that is an instance of l and that has o as last oid ($L1.o1.L2.o2. \dots Ln.o$)
 - The **target set** for label path l in a graph data s is the set of objects that can be reached in s via l .
→ Notation: $T(s,l)$
 - $L(s,l)$: set of label paths of s that have the same target set in s as l . (equivalent class)
- d is a **strong data guide** for s if for all label paths l of s it holds that $L(s,l) = L(d,l)$
- There is a 1-1-mapping between target sets in the data model and nodes in a strong data guide.

Strong data guide



Strong data guide

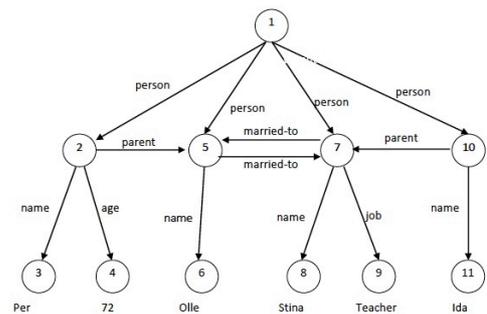


Strong data guide -- algorithm

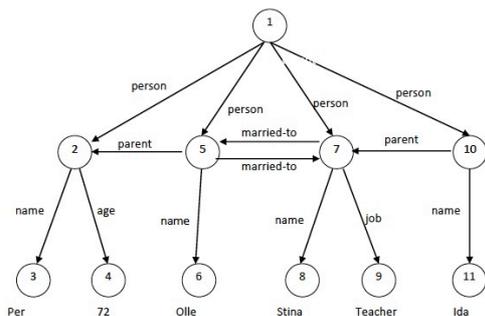
Implementation:

- Traverse data model depth-first.
- Each time you find a new target set for label path l , create a new object in the data guide.
- If the target set is already represented in the data guide, do not create a new object, but link to the existing object.

Strong data guide



Strong data guide



RestaurantGuide

