

CPT-S 415

Big Data

Yinghui Wu

EME B45

CPT-S 415

Big Data

Beyond Relational Data

- ✓ Introduction to XML
- ✓ XML basics
- ✓ DTD
- ✓ XML Schema
- ✓ XML Constraints

Semi-structured Data

- ✓ In many applications, data does not have a **rigidly** and **predefined** schema:
 - e.g., structured files, emails, scientific data, XML, JSON.
- ✓ Managing such data requires rethinking the design of components of a DBMS:
 - data model, query language, optimizer, storage system.
- ✓ The emergence of XML data underscores the importance of semi-structured data.

Main Characteristics

Schema is not what it used to be:

- ✓ not given in advance (often implicit in the data)
- ✓ descriptive, not prescriptive,
- ✓ partial,
- ✓ rapidly evolving,
- ✓ may be large (compared to the size of the data)

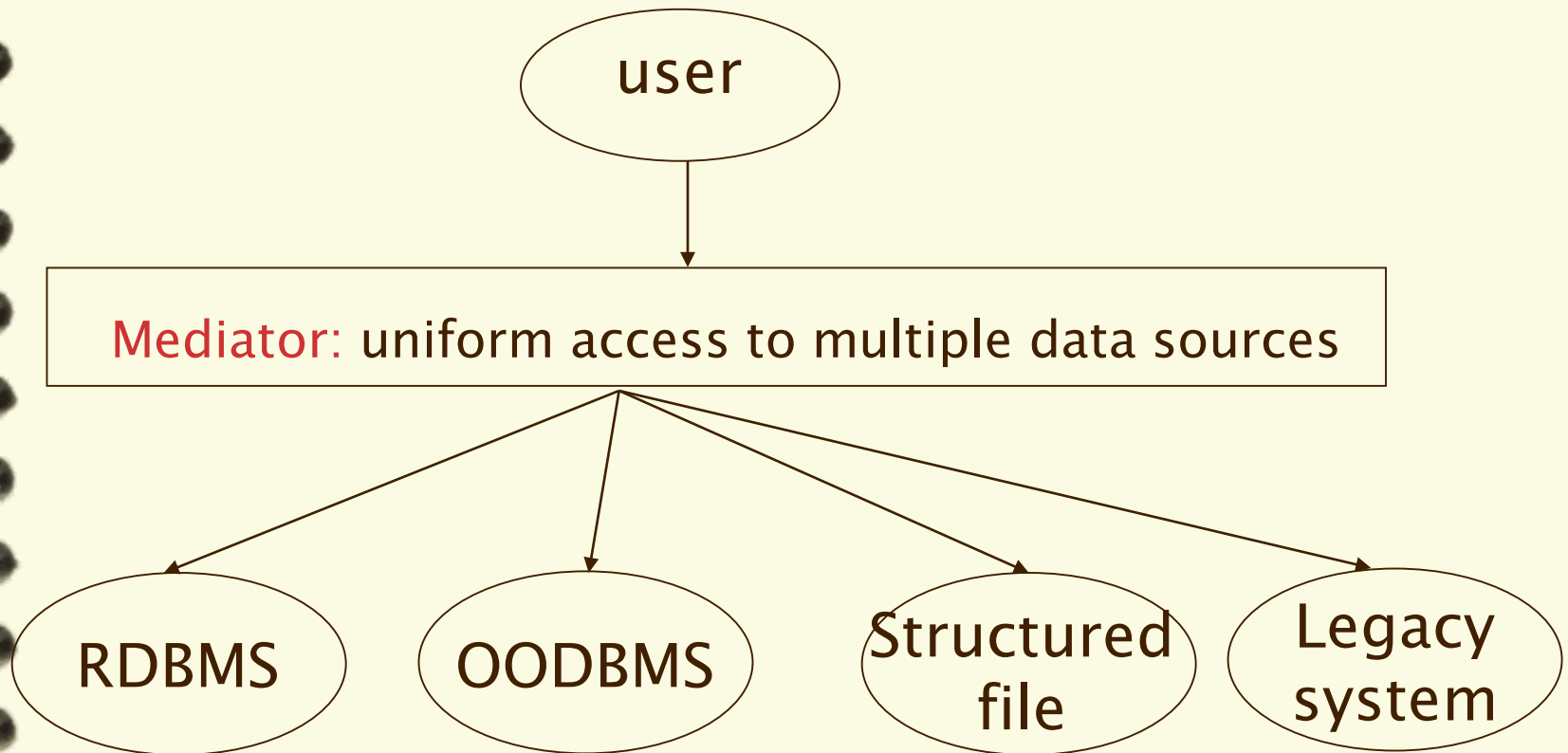
Types are not what they used to be:

- ✓ objects and attributes are not strongly typed
- ✓ objects in the same collection have different representations.

Example: XML

```
<bib>
  <book year="1995">
    <title> Database Systems </title>
    <author> <lastname> Date </lastname> </author>
    <publisher> Addison-Wesley </publisher>
  </book>
  <book year="1998">
    <title> Foundation for Object/Relational Databases
</title>
    <author> <lastname> Date </lastname> </author>
    <author> <lastname> Darwen </lastname> </author>
    <ISBN> <number> 01-23-456 </number> </ISBN>
  </book>
</bib>
```

Example: Data Integration



Each source represents data differently:
different data models, different schemas

Physical versus Logical Structure

- ✓ In some cases, data can be modeled in relational or object-oriented models, but **extracting the tuples is hard**
 - extracting data from HTML:
 - [Ashish and Knoblock, 97], [Hammer et al., 97], [Kushmerick and Weld, 97].
- ✓ **Semi-structured data:** when the data cannot be modeled naturally or usefully using a standard data model.

Managing Semi-structured Data

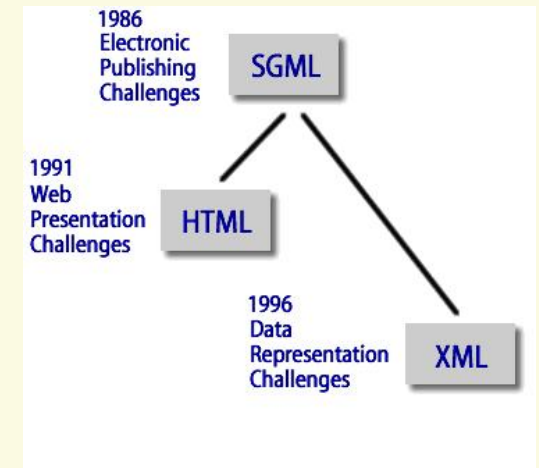
- ✓ **How do we model it?** (directed labeled graphs).
- ✓ **How do we query it?** (many proposals, all include regular path expressions).
- ✓ **Optimize queries?** (beginning to understand).
- ✓ **Store the data?** (looking for patterns)
- ✓ Integrity constraints, views, updates,...,
- ✓ We start with introduction of XML

History: SGML, HTML, XML

SGML: Standard Generalized Markup Language

-- Charles Goldfarb, ISO 8879, 1986

- ✓ DTD (Document Type Definition)
- ✓ powerful and flexible tool for structuring information, but
 - complete, generic implementation of SGML is difficult
 - tools for working with SGML documents are expensive
- ✓ two sub-languages that have outpaced SGML:
 - **HTML**: HyperText Markup Language (Tim Berners-Lee, 1991). Describing presentation.
 - **XML**: eXtensible Markup Language, W3C, 1998. Describing content.



From HTML to XML

HTML is good for presentation (**human friendly**), but does not help automatic data extraction by means of programs (**not computer friendly**).

Why? HTML tags:

- ✓ predefined and fixed
- ✓ describing display format, not the structure of the data.

<h3> George Bush **</h3>**

**** Taking CPTS 580-05 **** **
**

**** GPA: 1.5 **** **
**

<h3> Sloan 163 **</h3>**

**** Big Data ****

XML: a first glance

XML tags:

- ✓ user defined
- ✓ describing the structure of the data

```
<school>
```

```
  <student id = "011">
```

```
    <name>
```

```
      <firstName>George</firstName>  <lastName>Bush</lastName>
```

```
    </name>
```

```
    <taking> CPTS 580-05</taking>
```

```
    <GPA> 1.5 </GPA>
```

```
  </student>
```

```
  <course cno = "CPTS 580-05">
```

```
    <title> Big Data</title>
```

```
  </course>
```

```
</school>
```

XML vs. HTML

- ✓ user-defined new tags, describing structure instead of display
- ✓ structures can be arbitrarily nested (even recursively defined)
- ✓ optional description of its grammar (DTD) and thus validation is possible

What is XML for?

- ✓ The prime standard for data exchange on the Web
- ✓ A uniform data model for data integration

XML presentation:

- ✓ XML standard does not define how data should be displayed
- ✓ Style sheet: provide browsers with a set of formatting rules to be applied to particular elements
 - CSS (Cascading Style Sheets), originally for HTML
 - **XSL** (eXtensible Style Language), for XML

Tags and Text

- ✓ XML consists of tags and text
 - <course cno = "Eng 055">
 - <title> Spelling </title>
 - </course>
- ✓ tags come in pairs: markups
 - start tag, e.g., <course>
 - end tag, e.g., </course>
- ✓ tags must be properly nested
 - <course> <title> ... </title> </course> -- good
 - <course> <title> ... </course> </title> -- bad
- ✓ XML has only a single "basic" type: text, called **PCDATA** (Parsed Character DATA)

XML Elements

- ✓ Element: the segment between an start and its corresponding end tag
- ✓ subelement: the relation between an element and its component elements.

<person>

<name> Yinghui Wu </name>

<tel> (509) 335-7612</tel>

<email> yinghui@eecs.wsu.edu </email>

<email> yinghui@wsu.edu </email>

</person>

Nested Structure

- ✓ nested tags can be used to express various structures, e.g., “records”:

<person>

<name> Yinghui Wu </name>

<tel> (509) 335-7612 </tel>

<email> yinghui@eecs.wsu.edu </email>

<email> yinghui@wsu.edu </email>

</person>

- ✓ a list: represented by using the same tags repeatedly:

<person> ... </person>

<person> ... </person>

...

Ordered Structure

XML elements are ordered!

- ✓ How to represent **sets** in XML?
- ✓ How to represent an **unordered pair** (a, b) in XML?
- ✓ Can one directly represent the following in a relational database?
 - `<person> ... </person>`
`<person> ... </person> ...`
 - `<person>`
 - `<name> Yinghui Wu </name>`
 - `<tel> (509) 3357612 </tel>`
 - `<email> Yinghui@eecs.wsu.edu </email>`
 - `<email> Yinghui@wsu.edu </email>`
 - `</person>`

XML attributes

An start tag may contain attributes describing certain “properties” of the element (e.g., dimension or type)

```
<picture>
```

```
  <height dim="cm"> 2400</height>
```

```
  <width dim="in"> 96 </width>
```

```
  <data encoding="gif"> M05-+C$ ... </data>
```

```
</picture>
```

References (meaningful only when a DTD is present):

```
<person id = "011"  pal="012">
```

```
  <name> Barack Obama </name>
```

```
</person>
```

```
<person id = "012"  pal="011">
```

```
  <name> Hillary Clinton </name>
```

```
</person>
```

The “structure” of XML attributes

- ✓ XML attributes cannot be nested -- flat
- ✓ the names of XML attributes of an element must be unique.
one can't write `<person pal="Blair" pal="Saddam"> ...`

- ✓ XML attributes are **not ordered**

```
<person id = "011" pal="012">  
  <name> Barack Obama </name>  
</person>
```

is the same as

```
<person pal="012" id = "011">  
  <name> Barack Obama </name>  
</person>
```

- ✓ Attributes vs. subelements: unordered vs. ordered, and
 - attributes cannot be nested (flat structure)
 - subelements cannot represent references

Representing relational databases

A relational database for school:

student:

id	name	gpa
001	Joe	3.0
002	Mary	4.0
...

course:

cno	title	credit
331	DB	3.0
350	Web	3.0
...

enroll:

id	cno
001	331
001	350
002	331
...	...

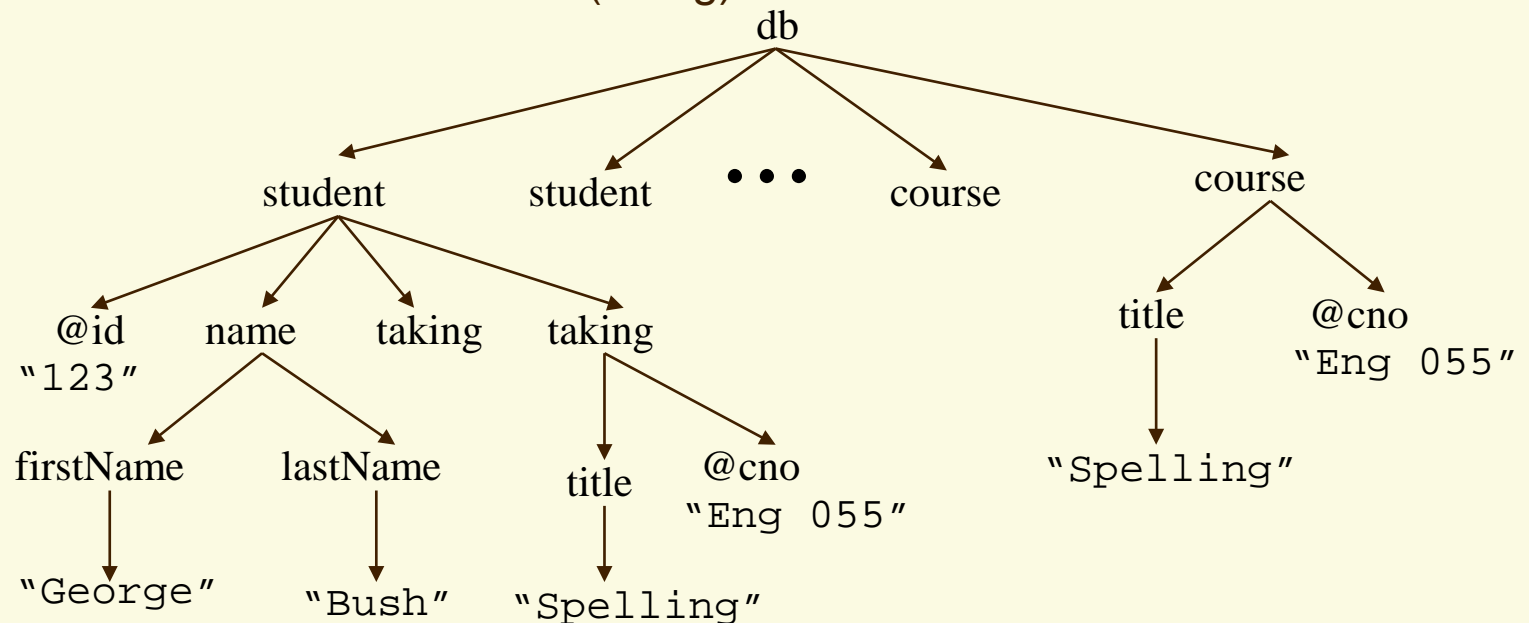
XML representation

```
<school>
  <student id="001">
    <name> Joe </name>          <gpa> 3.0 </gpa>
  </student>
  ...
  <course cno="331">
    <title> DB </title>        <credit> 3.0 </credit>
  </course>
  ...
  </course>
  <enroll>
    <id> 001 </id>             <cno> 331 </cno>
  </enroll>
  ...
</school>
```

The XML tree model

An XML document is modeled as a node-labeled ordered tree.

- ✓ **Element** node: typically internal, with a name (tag) and children (subelements and attributes), e.g., **student**, **name**.
- ✓ **Attribute** node: leaf with a name (tag) and text, e.g., **@id**.
- ✓ **Text** node: leaf with text (string) but without a name.



Introduction to XML

- ✓ XML basics
- ✓ DTDs
- ✓ XML Schema
- ✓ XML Constraints

Document Type Definition (DTD)

An XML document may come with an optional DTD – “schema”

```
<!DOCTYPE db [  
    <!ELEMENT db (book*)>  
    <!ELEMENT book (title, authors*, section*, ref*)>  
    <!ATTLIST book isbn ID #required>  
    <!ELEMENT section (text | section)*>  
    <!ELEMENT ref EMPTY>  
    <!ATTLIST ref to IDREFS #IMPLIED>  
    <!ELEMENT title #PCDATA>  
    <!ELEMENT author #PCDATA>  
    <!ELEMENT text #PCDATA>  
]>
```

Element Type Definition (1)

for each element type E, a declaration of the form:

$$\langle \text{!ELEMENT } E \text{ } P \rangle \quad E \rightarrow P$$

where P is a regular expression, i.e.,

$$P ::= \text{EMPTY} \mid \text{ANY} \mid \text{\#PCDATA} \mid E' \mid$$
$$P_1, P_2 \mid P_1 \mid P_2 \mid P? \mid P+ \mid P^*$$

- E': element type
- P1 , P2: concatenation
- P1 | P2: disjunction
- P?: optional
- P+: one or more occurrences
- P*: the Kleene closure

Element Type Definition (2)

- ✓ Extended context free grammar: `<!ELEMENT E P>`

Why is it called extended?

E.g., book → title, authors*, section*, ref*

- ✓ single root: `<!DOCTYPE db [...] >`

- ✓ subelements are **ordered**.

The following two definitions are different. Why?

`<!ELEMENT section (text | section)*>`

`<!ELEMENT section (text* | section*)>`

How to declare E to be an unordered pair (a, b)?

`<!ELEMENT E ((a, b) | (b, a)) >`

- ✓ **recursive** definition, e.g., section, binary tree:

`<!ELEMENT node (leaf | (node, node))`

`<!ELEMENT leaf (#PCDATA)>`

Element Type Definition (3)

- ✓ more on **recursive DTDs**

```
<!ELEMENT person (name, father, mother)>
```

```
<!ELEMENT father (person)>
```

```
<!ELEMENT mother (person)>
```

What is the problem with this? How to fix it?

- Attributes
- optional (e.g., father?, mother?)

```
<!ELEMENT person (name, father?, mother?)>
```

```
<!ELEMENT father (person) >
```

```
<!ELEMENT mother (person)>
```

Attribute declarations

General syntax:

```
<!ATTLIST element_name  
            attribute-name attribute-type default-declaration>
```

example: “keys” and “foreign keys”

```
<!ATTLIST    book  
            isbn ID #required>  
  
<!ATTLIST    ref  
            to IDREFS #implied>
```

Note: it is OK for several element types to define an attribute of the same name, e.g.,

```
<!ATTLIST    person name ID #required>  
<!ATTLIST    pet    name ID #required>
```

XML reference mechanism

- ✓ ID attribute: **unique** within **the entire document**.
 - An element can have at most one ID attribute.
 - No default (fixed default) value is allowed.
 - #required: a value must be provided
 - #implied: a value is optional
- ✓ IDREF attribute: its value must be some other element's **ID** value in the document.
- ✓ IDREFS attribute: its value is a set, each element of the set is the ID value of some other element in the document.

```
<person id="898" father="332" mother="336"  
        children="982 984 986">
```

Specifying ID and IDREF attributes

```
<!ATTLIST      person
                id      ID      #required
                father  IDREF   #implied
                mother  IDREF   #implied
                children IDREFS  #implied>
```

e.g.,

```
<person id="898" father="332" mother="336"
        children="982 984 986">
```

....

```
</person>
```

Valid XML documents

A **valid** XML document must have a DTD.

✓ It **conforms to** the DTD:

- elements conform to the grammars of their type definitions (nested only in the way described by the DTD)
- elements have all and only the attributes specified by the DTD
- ID/IDREF attributes satisfy their constraints:
 - ID must be distinct
 - IDREF/IDREFS values must be existing ID values

Introduction to XML

- ✓ XML basics
- ✓ DTDs
- ✓ XML Schema
- ✓ XML Constraints

DTDs vs. schemas (types)

- ✓ By the database (or programming language) standard, XML DTDs are rather weak specifications.
 - Only one base type -- PCDATA.
 - No useful “abstractions”, e.g., unordered records.
 - No sub-typing or inheritance.
 - IDREFs are not typed or scoped -- you point to something, but you don’t know what!
- ✓ XML extensions to overcome the limitations.
 - Type systems: XML-Data, XML-Schema, SOX, DCD
 - Integrity Constraints

XML Schema

Official W3C Recommendation

A rich type system:

- ✓ Simple (atomic, basic) types for both element and attributes
- ✓ Complex types for elements
- ✓ Inheritance
- ✓ Constraints
 - key
 - keyref (foreign keys)
 - uniqueness: “more general” keys
- ✓ . . .

See www.w3.org/XML/Schema for the standard and much more

Atomic types

- ✓ string, integer, boolean, date, ...,
- ✓ enumeration types
- ✓ restriction and range [a-z]
- ✓ list: list of values of an atomic type, ...

Example: define an element or an attribute

```
<xs:element name="car" type="carType">  
<xs:attribute name="car" type="carType">
```

Define the type:

```
<xs:simpleType name="carType">  
  <xs:restriction base="xs:string">  
    <xs:enumeration value="Audi">  
    <xs:enumeration value="BMW">  
  </xs:restriction>  
</xs:simpleType>
```

Complex types

- ✓ Sequence: “record type” – ordered
- ✓ All: record type – unordered
- ✓ Choice: variant type
- ✓ Occurrence constraint: maxOccurs, minOccurs
- ✓ Group: mimicking parameter type to facilitate complex type definition
- ✓ Any: “open” type – unrestricted
- ✓ ...

Example

A complex type for publications:

```
<xs:complexType name="publicationType">
  <xs:sequence>
    <xs:choice>
      <xs:group ref="journalType">
        <xs:element name="conference" type="xs:string"/>
      </xs:choice>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="author" type="xs:string"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

Example (cont'd)

```
<xs:group name="journalType">  
  <xs:sequence>  
    <xs:element name="name" type="xs:string"/>  
    <xs:element name="volume" type="xs:integer"/>  
    <xs:element name="number" type="xs:integer"/>  
  </xs:sequence>  
</xs:group>
```

Inheritance -- Extension

Subtype: extending an existing type by including additional fields

```
<xs:complexType name="datedPublicationType">
  <xs:complexContent>
    <xs:extension base="publicationType">
      <xs:sequence>
        <xs:element name="isbn" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="publicationDate" type="xs:date"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Inheritance -- Restriction

Supertype: restricting/removing certain fields of an existing type

```
<xs:complexType name="anotherPublicationType">
  <xs:complexContent>
    <xs:restriction base="publicationType">
      <xs:sequence>
        <xs:choice>
          <xs:group ref="journalType">
            <xs:element name="conference" type="xs:string"/>
          </xs:choice>
          <xs:element name="author" type="xs:string"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
```

Removed title

Introduction to XML

- ✓ XML basics
- ✓ DTDs
- ✓ XML Schema
- ✓ XML Constraints

Keys and Foreign Keys

Example: school document

```
<!ELEMENT db (student+, course+) >
<!ELEMENT student (id, name, gpa, taking*)>
<!ELEMENT course (cno, title, credit, taken_by*)>
<!ELEMENT taking (cno)>
<!ELEMENT taken_by (id)>
```

- ✓ keys: locating a specific object, an invariant connection from an object in the real world to its representation

student.@id → student, course.@cno → course

- ✓ foreign keys: referencing an object from another object

taking.@cno \subseteq course.@cno, course.@cno → course
taken_by.@id \subseteq student.@id, student.@id → student

Constraints are important for XML

- ✓ Constraints are a fundamental part of the semantics of the data; **XML may not come with a DTD/type** – thus constraints are often the only means to specify the semantics of the data
- ✓ Constraints have proved useful in
 - semantic specifications: obvious
 - query optimization: effective
 - database conversion to an XML encoding: a must
 - data integration: information preservation
 - update anomaly prevention: classical
 - normal forms for XML specifications: “BCNF”, “3NF”
 - efficient storage/access: indexing,
 - ...

The limitations of the XML standard (DTD)

ID and IDREF attributes in DTD vs. keys and foreign keys in RDBs

✓ Scoping:

- ID unique **within the entire document** while a key needs only to uniquely identify a tuple **within a relation**
- IDREF **untyped**: one has no control over what it points to -- you point to something, but you don't know what it is!

```
<student id="01" name="Hillary" taking="CPTS580"/>
```

```
<student id="02" name="Bush" taking="CPTS580  
05"/>
```

```
<course id="CPTS580"/>
```

The limitations of the XML standard (DTD)

- ✓ keys can be **multi-valued**, while IDs must be single-valued (unary)

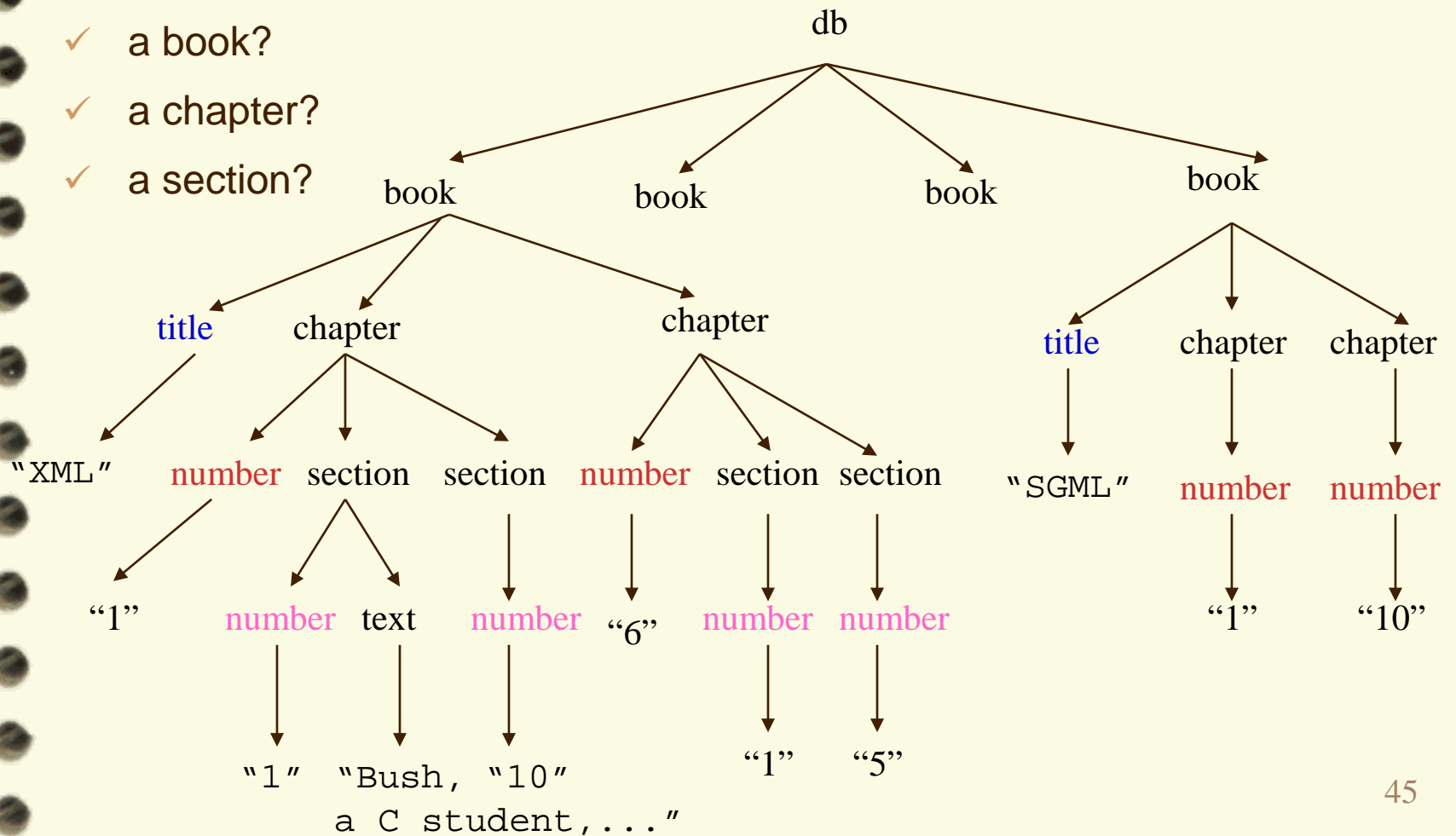
enroll (sid: string, cid: string, grade:string)

- ✓ a relation may have multiple keys, while an element can have at **most one** ID (primary)
- ✓ ID/IDREF can only be defined **in a DTD**, while XML data may not come with a DTD/schema
- ✓ ID/IDREF, even relational keys/foreign keys, fail to capture the semantics of **hierarchical data** – will be seen shortly

New challenges of hierarchical XML data

How to identify in a document

- ✓ a book?
- ✓ a chapter?
- ✓ a section?



Path expressions

Path expression: navigating XML trees

A simple **path language**:

$$q ::= \varepsilon \mid l \mid q/q \mid //$$

- ✓ ε : empty path
- ✓ l : tag
- ✓ q/q : concatenation
- ✓ $//$: descendants and self – recursively descending downward

To overcome the limitations

Absolute key: $(Q, \{P_1, \dots, P_k\})$

- ✓ target path Q : to identify a **target set** $[[Q]]$ of nodes on which the key is defined (vs. relation)
- ✓ a set of **key paths** $\{P_1, \dots, P_k\}$: to provide an identification for nodes in $[[Q]]$ (vs. key attributes)
- ✓ semantics: for any two nodes in $[[Q]]$, if they *have all the key paths* and agree on them up to **value equality**, then they must be the same node (**value equality** and **node identity**)

(//student, { @id })

(//student, { //name })

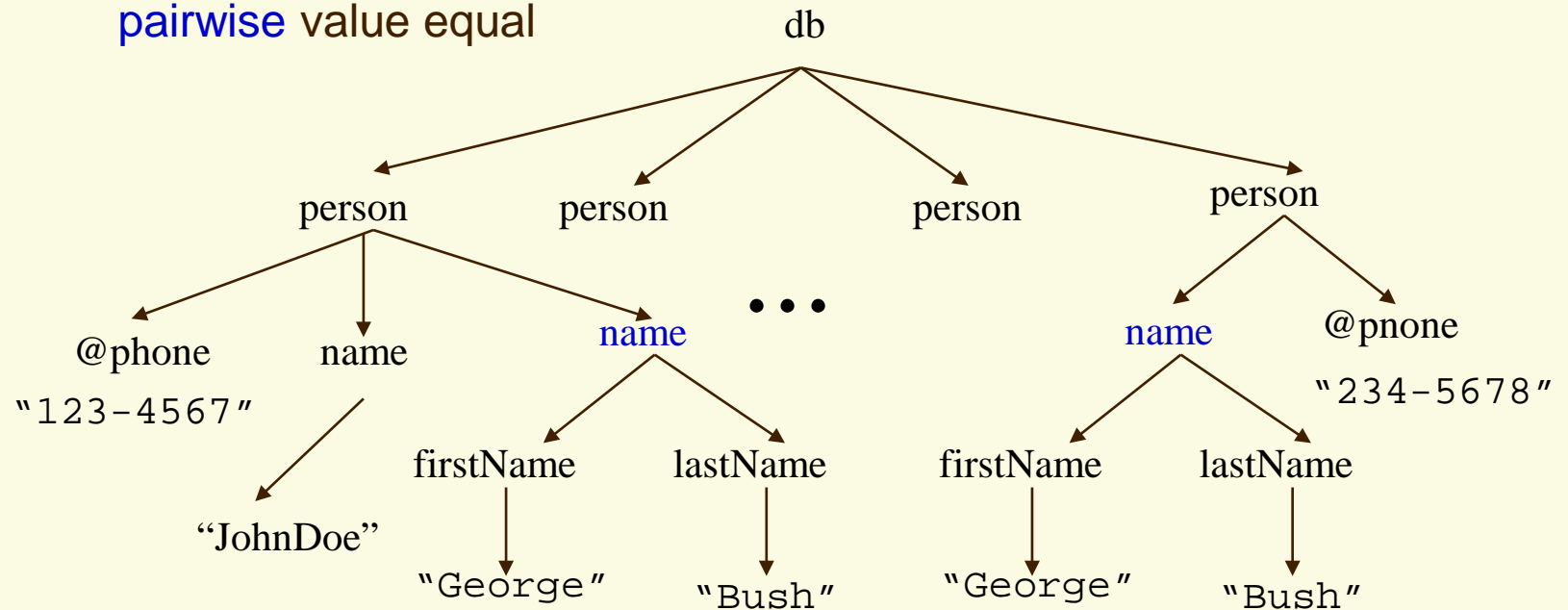
(//enroll, { @id, @cno })

(//, { @id })

Value equality on trees

Two nodes are **value equal** iff

- ✓ either they are text nodes (PCDATA) with the same value;
- ✓ or they are attributes with the same tag and the same value;
- ✓ or they are elements having the same tag and their children are **pairwise** value equal

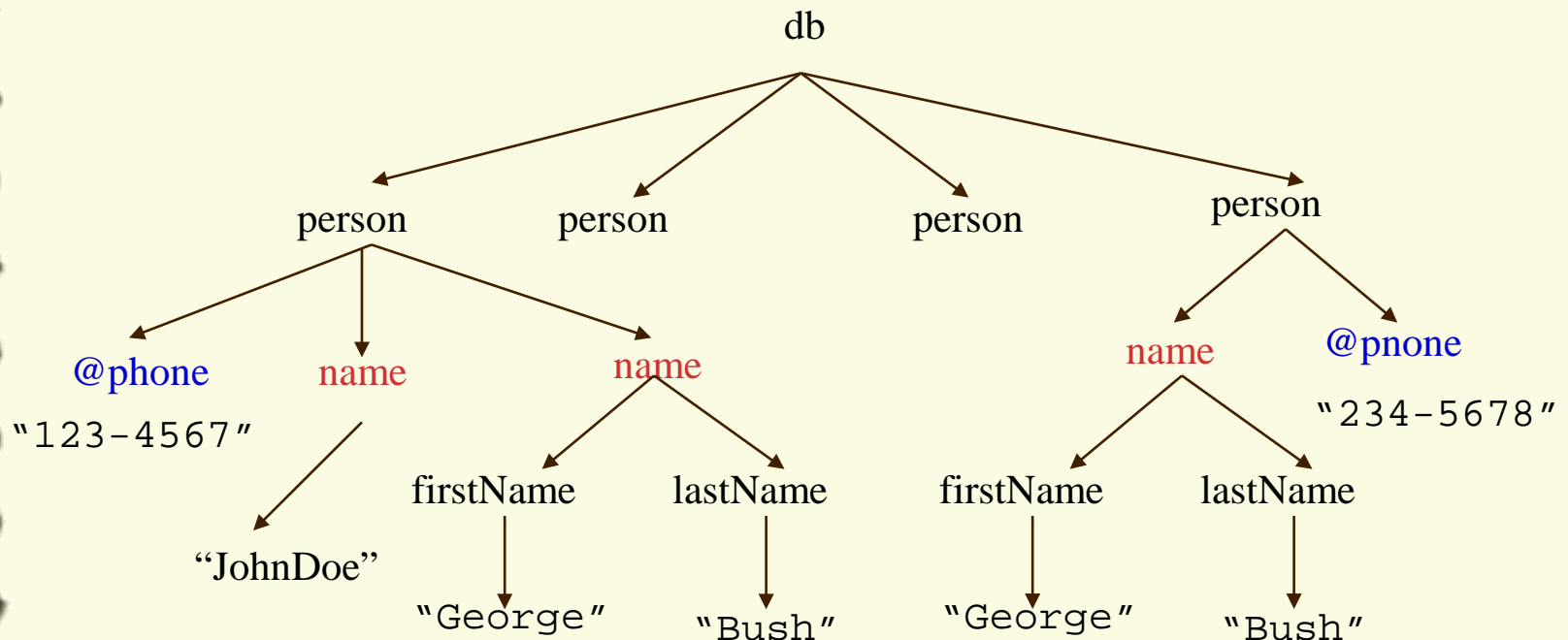


Capturing the semistructured nature of XML data

- ✓ independent of types – no need for a DTD or schema
- ✓ no structural requirement: tolerating **missing/multiple** paths

(//person, {name})

(//person, {name, @phone})



Relative constraints

Relative key: (Q, K)

- ✓ path Q identifies a set $[[Q]]$ of nodes, called the **context**;
- ✓ $k = (Q', \{P_1, \dots, P_k\})$ is a key on sub-documents rooted at nodes in $[[Q]]$ (relative to Q).

Example.

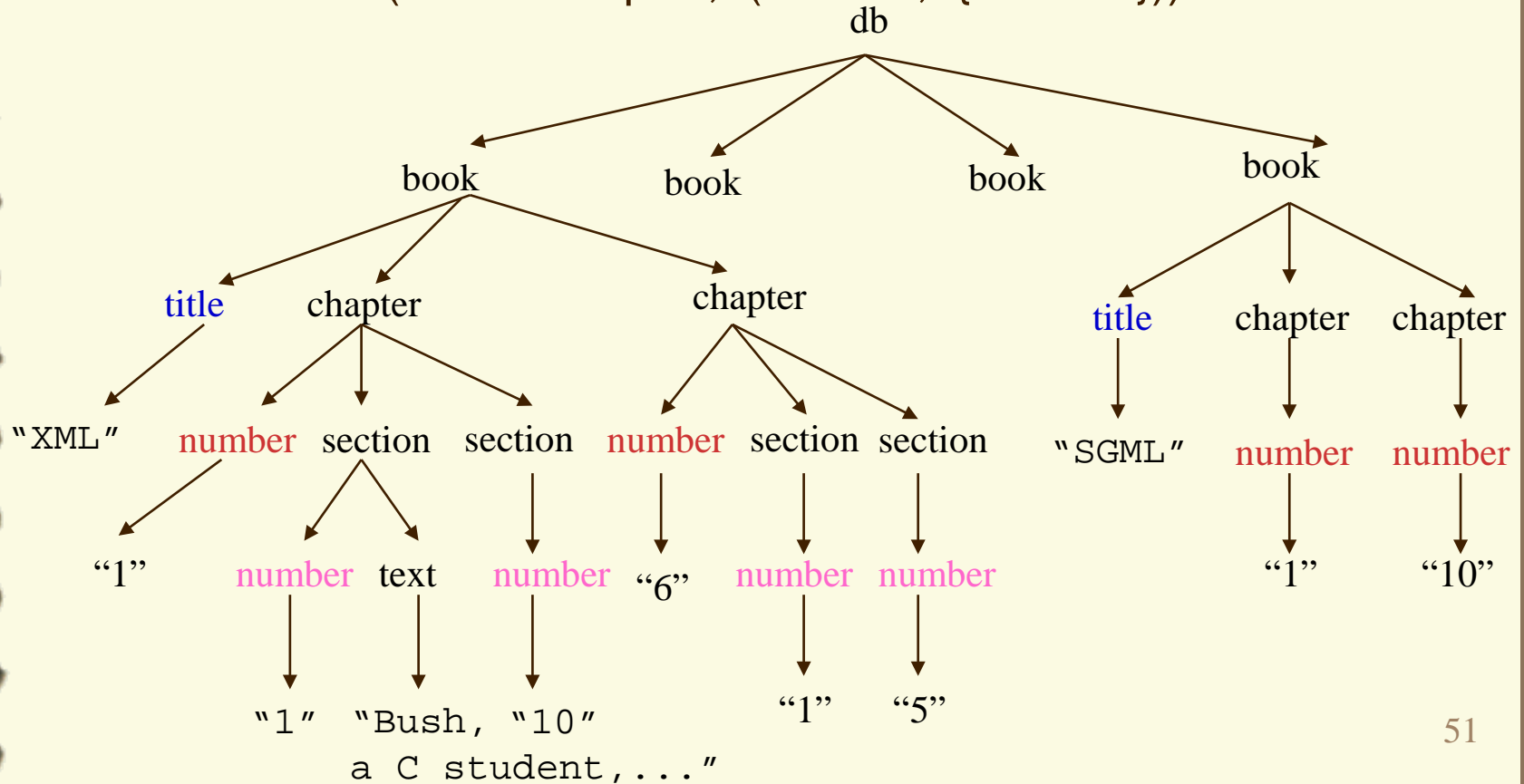
(//book, (chapter, {number}))

(//book/chapter, (section, {number}))

(//book, {title}) -- absolute key

Examples of XML constraints

absolute (//book, {title})
 relative (//book, (chapter, {number}))
 relative (//book/chapter, (section, {number}))



Absolute vs. relative keys

- ✓ Absolute keys are a special case of relative keys:
(Q, K) when Q is the empty path
- ✓ Absolute keys are defined on the entire document, while relative keys are scoped within the context of a sub-document
- ✓ Important for hierarchically structured data: XML, scientific databases, ...

absolute (//book, {title})

relative (//book, (chapter, {number}))

relative (//book/chapter, (section, {number}))

XML keys are more **complex** than relational keys!

Summary and Review

- ✓ XML is a prime data exchange format.
- ✓ DTD provides useful syntactic constraints on documents.
- ✓ XML Schema extends DTD by supporting a rich type system
- ✓ Integrity constraints are important for XML, yet are nontrivial

Exercise:

- ✓ Design a DTD and an XML Schema to represent student, enroll and course relations. Give necessary XML constraints
- ✓ Convert student and course relations to an XML document based on your DTD/Schema
- ✓ Is XML capable of modeling an arbitrary relational/object-oriented database?
- ✓ Take a look at XML interface: DOM (Document-Object Model), SAX (Simple API for XML). What are the main differences?
- ✓ Study tutorials for XPath, XSLT and XQuery