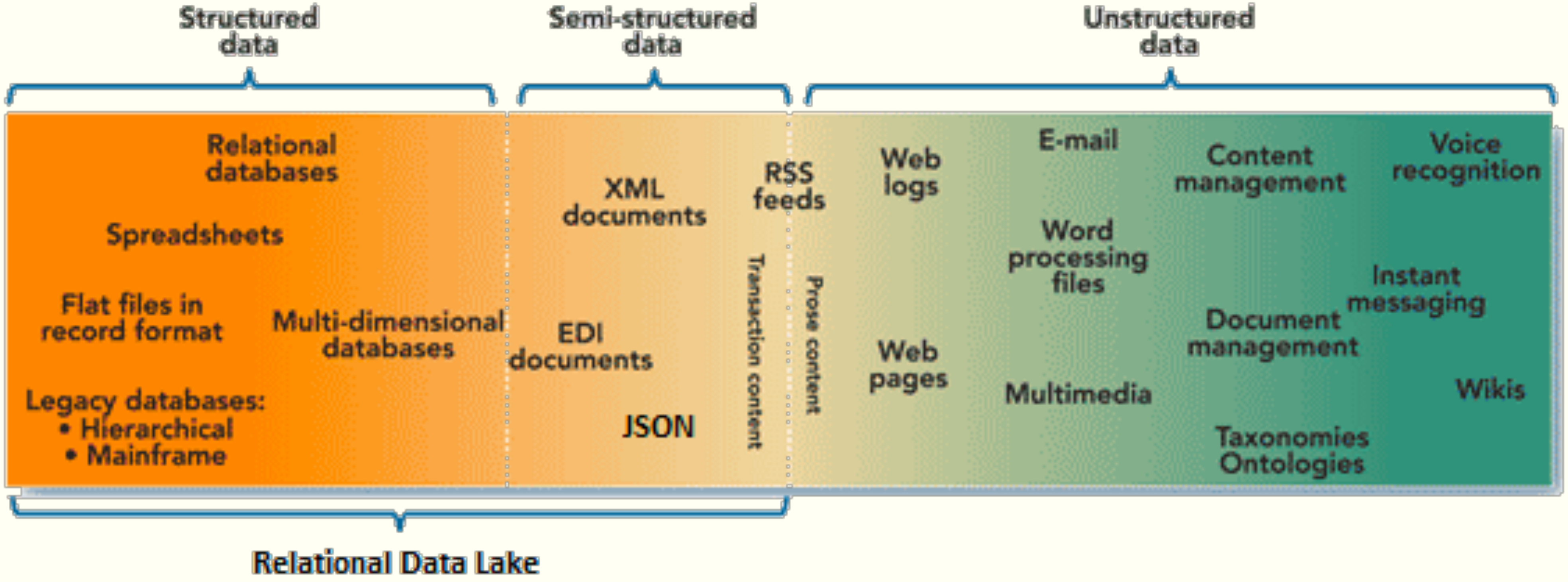
CptS 415 Big Data

Introduction to XML

Srini Badri

Acknowledgements: Tinghui Wang

Where Are We?


Structured data
Semi-structured data
Unstructured data
Relational databases
Spreadsheets
Flat files in record format
Multi-dimensional databases
Legacy databases:
• Hierarchical
• Mainframe
XML documents
EDI documents
JSON
RSS feeds
Transaction content
Prose content
Web logs
Web pages
E-mail
Word processing files
Multimedia
Content management
Document management
Taxonomies Ontologies
Voice recognition
Instant messaging
Wikis
Relational Data Lake

Beyond Relational Data

- Introduction to XML
- XML Basics
- DTD
- XML Schema
- XML Constraints

Semi-Structured Data

- Im many applications, data does not have a rigidly and predefined schema:
    - 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
- XML data underscores the importance of semi-structured data

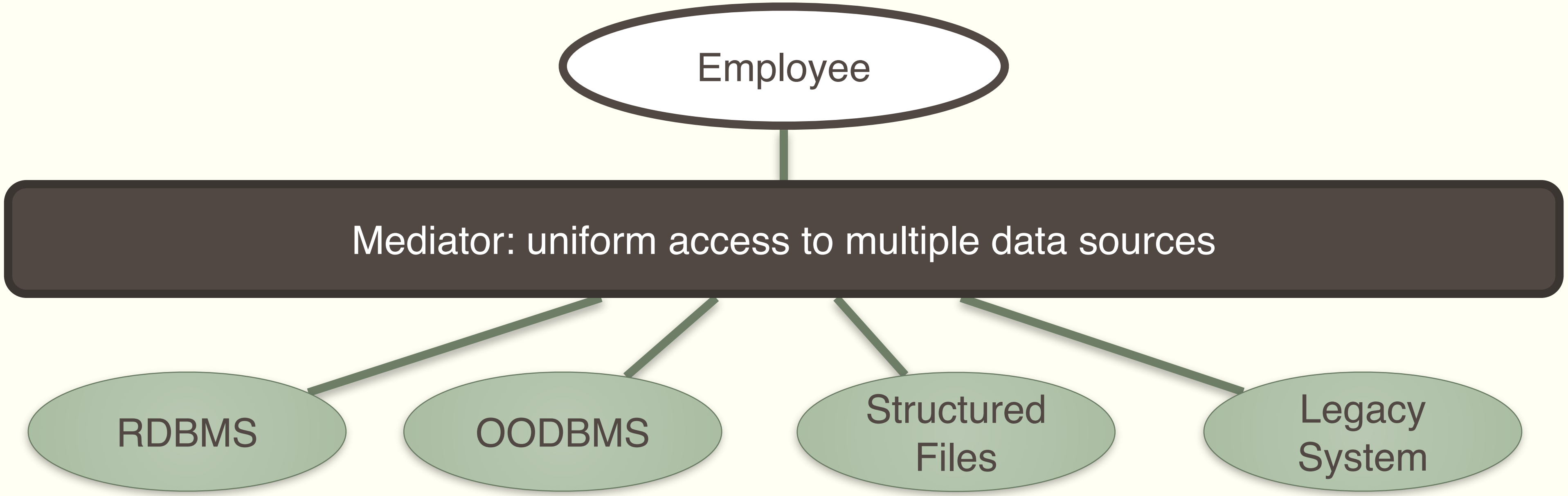
Main Characteristics

- 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



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

Physical Vs. Logical Structure

- In some cases, data can be modeled in relational or objected oriented models
- Extracting 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 graph
- How do we query it?
  - Many proposals, all include regular path expressions
- How to optimize queries?
- How to store the data?
- Integrity constraints, views, updates, …

History: SGML, HTML, XML

- Standard Generalized Markup Language (SGML)
  - Charles Goldfarb, ISO 8879, 1986
- Powerful and flexible tool for structuring information, but
  - Complete, generic implementation of SGML is difficult
  - Tools that works with SGML documents are expensive
- Two sub-languages that have out-paced SGML:
  - HTML: HyperText Markup Language (Tim Berners-Lee, 1991), Describing presentation.
  - XML: eXtensible Markup Language, W3C , 1998. Describing contents.

Video Script Example: Ordinary Text Version

Schindler: Filing; billing; keeping track of my appointments; typing, obviously. How is your typing?
Woman: Uh... all right.
Schindler: Please. [Motions that she sit down]
Narrator: As painters refurbish his factory office, Schindler has a young woman sit behind a typewriter. Schindler folds his arms and stares at her smooth, pretty face. [Types away] Another girl, a brunette, sits at the typewriter and smiles. Now, a tall blonde. Schindler kindly slides back the return carriage for her. She grins. Next, a young lady squints and leans close to the paper. Another smiling blonde. A thin, wavy-haired brunette with a coy smile. Schindler leans on the desk and gazes at a stunning young woman, who returns his unwavering stare. Now, a despondently-slouching Schindler as a gruff middle-aged woman types away, a cigarette dangling from her lips.
Stern: You need a secretary. Pick one.
Schindler: I don't know how. They're all so... *qualified*.
Stern: You have to choose.
Narrator: Outside, Schindler poses with 20 beautiful secretaries.
Photographer: Big smile, big smile!
Narrator: A photographer takes a picture. [Scene changes]
Now, another scene:
[Pfefferberg knocks on doorglass]
Pfefferberg: Herr Direktor?

Video Script Example: Structure

- Original dialogue
- What the DVS narrator says
- Notation of sound effects and action
- Emphasis
- Pauses (notated by an ellipsis, ...)
- Manner of speaking: Quietly, to someone, etc.
- Exact pacing, which in a real example we would indicate by precise timecodes

Video Script Example: Structure Markup

- For dialogue,
  - <dialogue NAME> </dialogue NAME>
  - where NAME is the character's name: <dialogue Schindler> <dialogue Pfefferberg>
- For descriptions, something like
  - <description> </description>
- For sound effects,
  - <sound effect> </sound effect>;
- For action,
  - <action> </action>
- For emphasis
  - the HTML tag <em> </em>
- For pauses
  - <pause> (doesn't need an off tag)
- For manner,
  - <manner SPECIFICS> where we fill the SPECIFICS in, like <manner shouting> <manner to himself> <manner mumbling>

Video Script Example: SGML

<dialogue Schindler>Filing; billing; keeping track of my appointments; typing, obviously. How is your typing?</dialogue Schindler>

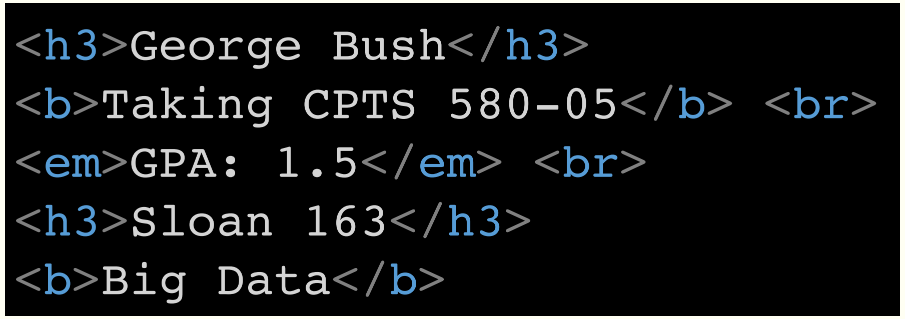
<dialogue Woman>Uh<pause> all right. </dialogue Woman>

<dialogue Schindler>Please.</dialogue Schindler> <action>Motions that she sit down</action>

<description>As painters refurbish his factory office, Schindler has a young woman sit behind a typewriter. Schindler folds his arms and stares at her smooth, pretty face. <action>Types away</action> Another girl, a brunette, sits at the typewriter and smiles. Now, a tall blonde. Schindler kindly slides back the return carriage for her. She grins. Next, a young lady squints and leans close to the paper. Another smiling blonde. A thin, wavy-haired brunette with a coy smile. Schindler leans on the desk and gazes at a stunning young woman, who returns his unwavering stare. Now, a despondently-slouching Schindler as a gruff middle-aged woman types away, a cigarette dangling from her lips.</description>

HTML

- HTML is good for presentation (human friendly)
- Does not help with automatic data extraction by means of programs (not computer friendly)
- HTML Tags:
  - Predefined and fixed
  - Describing display format, not the structure of data



```
<h3>George Bush</h3>
<b>Taking CPTS 580-05</b> <br>
<em>GPA: 1.5</em> <br>
<h3>Sloan 163</h3>
<b>Big Data</b>
```

XML

- XML Tags:
  - User defined
  - Describing the structure of the data

```
<school>
  <student id = “011”>
    <name>
      <firstName>George</firstName>
      <lastName>Bush</lastName>
    </name>
    <taking>CPTS 415</taking>
    <GPA>3.5</GPA>
  </student>
  <course cno = “CPTS 415”>
    <title>Big Data</title>
  </course>
</school>
```

XML: Features

- User-defined 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

XML

- What is it for?
  - The primary 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

Tag and Text

- XML consists of tags and text

```
<course cno = “CPTS 415”>
  <title>Big Data</title>
</course>
```

- Tags come in pairs: markups
  - Start tag
  - Stop tag
- Tags must be properly nested
- XML only has a single “basic” type: text (PCDATA)
  - Parsed Character DATA

XML Elements

- Element:
  - The segment between an stat and its corresponding end tag
- Subelement:
  - The relation between an element and its component elements

```
<student id = “011”>
  <name>
    <firstName>George</firstName>
    <lastName>Bush</lastName>
  </name>
  <taking>CPTS 415</taking>
  <GPA>3.5</GPA>
</student>
```

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?

XML Attributes

- A start tag may contain attributes describing certain properties of the element
- References (meaningful only when a DTD is present)

The Structure of XML Attributes

- XML Attributes cannot be nested – flat
- The names of XML attributes of an element must be unique
- XML attributes are not ordered
- Attributes Vs. Sub-elements:
  - Unordered vs Ordered

Representing Relational Databases

- A Relational Database for School

Student

ID	Name	GPA
001	Joe	3.0
002	Mary	4.0
…	…	…

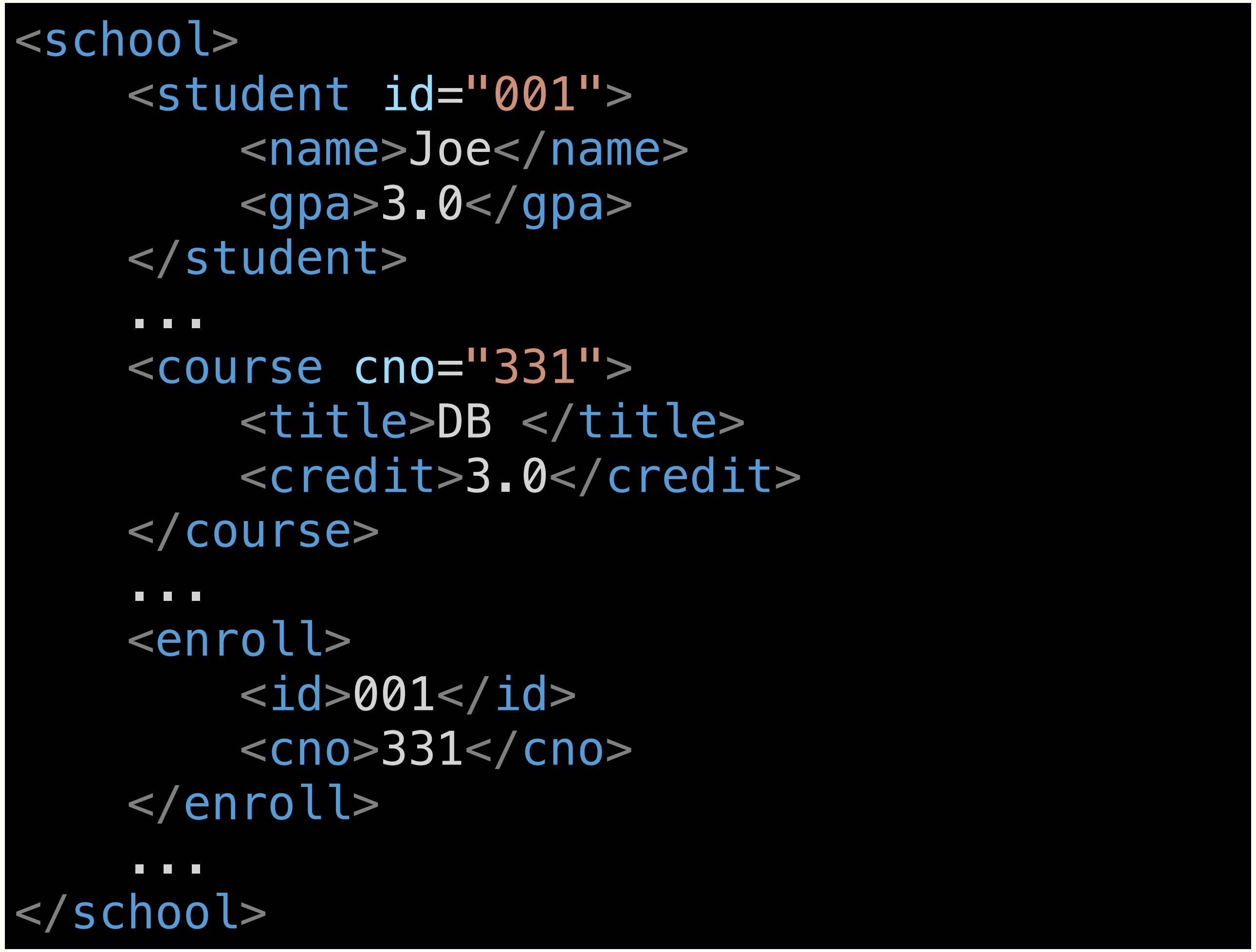
Course

CNO	TITLE	CREDIT
331	DB	3.0
350	Web	4.0
…	…	…

Enrollment

ID	CNO
001	331
002	350
002	331
…	…

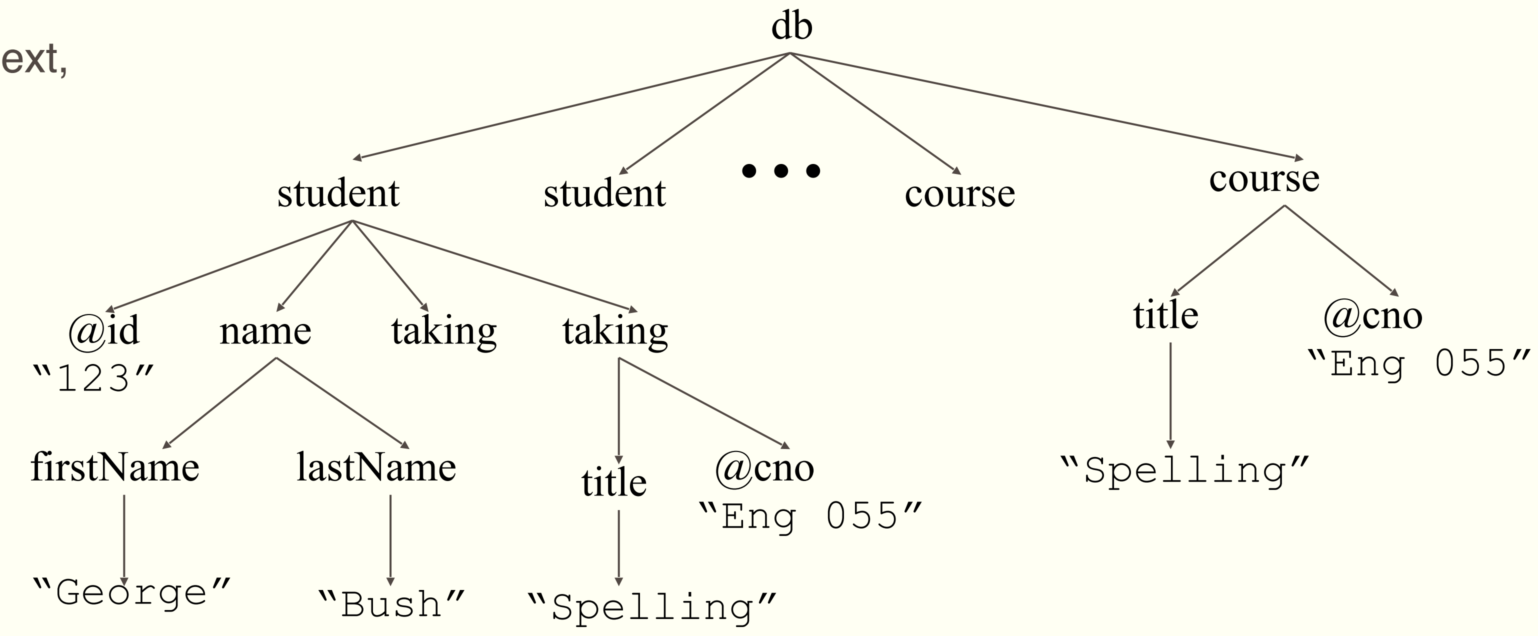
Example: XML Representation



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

XML Tree Model

- An XML document is modeled as a node-labeled ordered tree
- Element Node:
  - Typically internal, with a name (tag) and children (sub-elements 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



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

- For each element type E, a declaration of the form

```
<!ELEMENT E P>
```

- Where P is a regular expression, i.e.
- P ::= EMPTY | ANY | #PCDATA | E’ | (P1,P2) | (P1|P2) | P? | P+ | P*
    - E’: Element Type
    - P1, P2: Concatenation
    - P1 | P2: disjunction
    - P?: optional
    - P+: one or more occurrences
    - P*: The Kleene closure

Element Type Definition (Cont’d)

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

```
<!ELEMENT book (title, authors*, section*, ref*)>
```

- Single root: `<!DOCTYPE db []>`
- Sub-elements are ordered!

```
<!ELEMENT section (text | section)*>
<!ELEMENT section (text* | section*)>
<!ELEMENT section ((a, b)|(b, a))>          unordered
<!ELEMENT node (leaf | (node, node))>       recursive
<!ELEMENT leaf (#PCDATA)>
```

Element Type Definition (Cont’d)

- Another example on recursive DTD

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

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

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

Attribute Declarations

- General syntax

```
<!ATTLIST E_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

```
<!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.

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

Example: ID, IDREF and IDREFS attributes

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

```
<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 definition

      (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

DTD Vs. Schemas (Types)

- By the database (or programming language) standard, XML DTDs are rather week specification
  - Only one base type: #PCDATA
  - No useful abstractions, e.g. unordered records
  - No sub-typing or inheritance
  - IDREFs are not typed or scoped – point to something, but you do not 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 details

Atomic Types

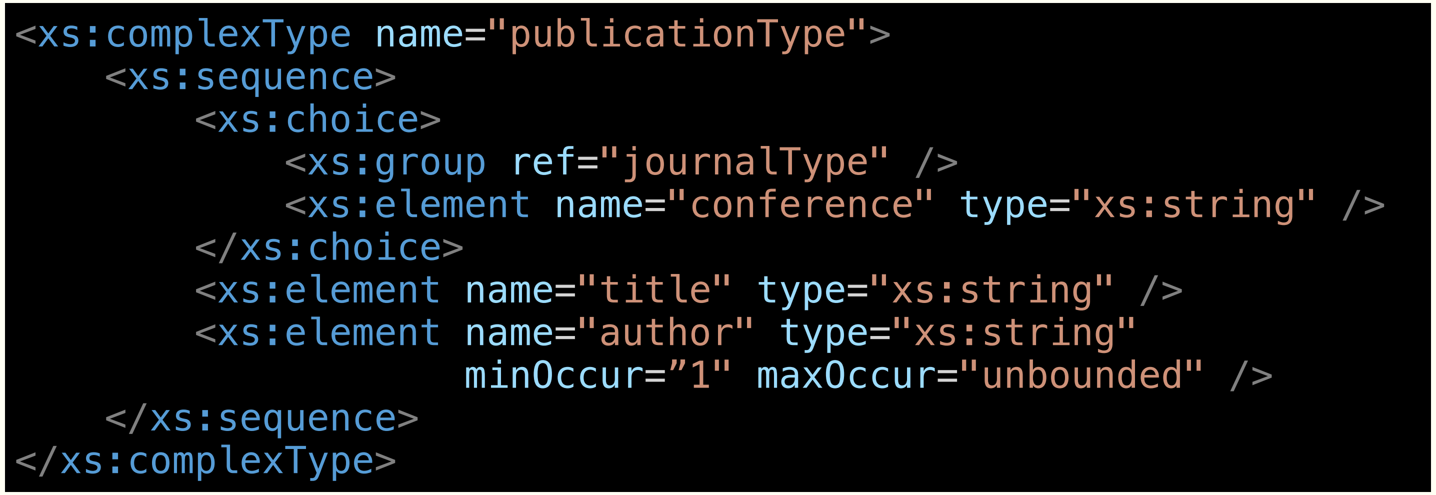
- String, integer, Boolean, date, …
- Enumeration types
- Restriction and range, e.g. [a-z]
- List: list of values of an atomic type

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

```
<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



```
<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"
                    minOccur=”1" maxOccur="unbounded" />
    </xs:sequence>
</xs:complexType>
```

Extension: Inheritance

- 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>
```

Extension: Inheritance (Cont’d)

- 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"
                            minOccur="1" maxOccur="unbounded" />
            </xs:sequence>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>
```

XML Constraints: Keys and Foreign Keys

```
<!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 collection from an object isn 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

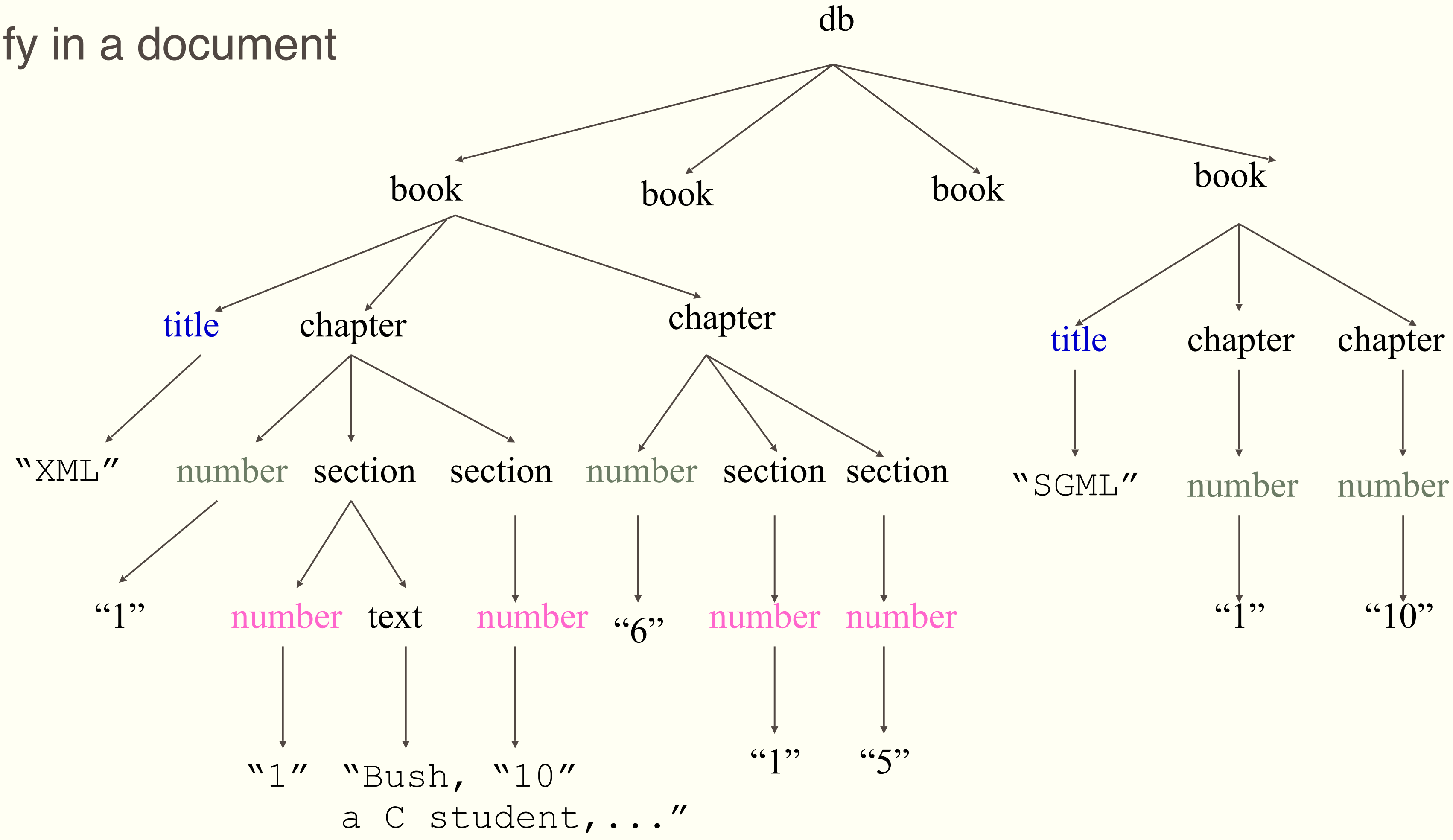
taken_by.@id $\subseteq$ student.@id; student.@id → student

Limitations of the XML standard (DTD)

	XML DTD	Relational DBMS
ID Vs. Key	Unique within the entire document	Uniquely identify a tuple within a relation
	Single-valued (unary)	Can be multi-valued
IDREF Vs. Foreign Key	Untyped, one has no control over what it points to	Primary key of a tuple in another relation
	Failed to capture the semantics of hierarchical data	

New Challenges of Hierarchical XML Data

- How to identify in a document
  - A book?
  - A chapter?
  - A section?



db
book
book
book
book
title
chapter
chapter
title
chapter
chapter
"XML"
number
section
section
number
section
section
"SGML"
number
number
"1"
number
text
number
"6"
number
number
"1"
"10"
"1"
"Bush, a C student,..."
"10"
"1"
"5"