Introduction to XML and Relational Databases

Spring 2005

Document Type Definition

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

```xml
<!DOCTYPE db [ 
  <!ELEMENT  db  (book*)> 
  <!ELEMENT  book   (title,  chapter*, ref*)> 
  <!ATTLIST  book  isbn   ID   #required> 
  <!ELEMENT  chapter  (number, section*) > 
  <!ELEMENT    section  (number, (text | section)*)> 
  <!ELEMENT  ref  EMPTY> 
  <!ATTLIST  ref  to    IDREFS   #implied> 
  <!ELEMENT  title  #PCDATA> 
  <!ELEMENT  text  #PCDATA> 
]
```
What is a DTD?

- A DTD constraints the structure of an XML document, and may help us formulate/optimize our queries.
- There is a relationship between a DTD and a databases schema or a type/class declaration of a program, but it is not close – hence the need for additional "typing" systems, such as XML Schema.
- A DTD is a syntactic specification. Its connection with any "conceptual" model may be quite remote.
- DTDs do not act like type systems for XQuery, XPath or XSLT. You can “validate” your XML documents, but that does not mean that your programs are checked for type errors.
Element Type Definition (1)

For each element type \( E \), a declaration of the form:

\[
<!ELEMENT \ E \ P> 
\]

where \( P \) is a regular expression, i.e.,

\[
P ::= \text{EMPTY} | \text{ANY} | \#PCDATA | E' | P_1 | P_2 | P_1 \cdot P_2 | P_1 \lor P_2 | P_1 ? | P_1 + | P_1 *
\]

- \( E' \): element type
- \( P_1, P_2 \): concatenation
- \( P_1 \lor P_2 \): disjunction
- \( P_1 ? \): optional
- \( P_1 + \): one or more occurrences
- \( P_1 * \): the Kleene closure

Element Type Definition (2)

- Extended context free grammar: <!ELEMENT \ E \ P>
  
  Why is it called extended?

  E.g., <!ELEMENT \ book \ (title, chapter*, ref*)> 

- single root: <!DOCTYPE \ db \ [ \ … \ ] > 

- subelements are ordered.

  The following two definitions are different. Why?

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

- 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?)

exercise

✓ What is the problem with the following?

<!ELEMENT  person (name,
    person?, /*father
      person? /* mother)

> 

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

<!ELEMENT  E ((a, b) | (b, a))>
Element Type Definition (4)

✓ EMPTY element:

<!ELEMENT ref EMPTY>
<!ATTLIST ref to IDREFS #implied>
observe that it has attributes

✓ ANY: may contain any content

<!ELEMENT generic ANY>

✓ mixed content

<!ELEMENT section (#PCDATA | section)>
exercise

What is the problem with the following?

```xml
<!ELEMENT student (id, name, gpa)>
<!ELEMENT name (first-name, last-name)>
...
<!ELEMENT course (cno, name, credit)>
<!ELEMENT name (PCDATA)>
```

Attribute declarations (1)

General syntax:

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

example: “keys” and “foreign keys”

```xml
<!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.,

```xml
<!ATTLIST person name ID #required>
<!ATTLIST pet name ID #required>
```
Attribute declarations (2)

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

✓ attribute types:
  – CDATA
  – ID, IDREF, IDREFS
  – …

✓ default declarations:
  – #required, #implied
  – "default value", #fixed "default value"

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>
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 already 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">
```

Keys and Foreign Keys

Example: school document

```xml
<!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 – within a relation
  - student.@id → student
  - course.@cno → course
- foreign keys: referencing an object from another object
  - taking.@cno ⊆ course.@cno
  - course.@cno → course
  - taken_by.@id ⊆ student.@id
  - student.@id → student
The limitations of ID/IDREF

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

✓ Scoping:

- ID unique within the entire document (like oids), 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!

```xml
@student id="01" name="Saddam" taking="CS2"/>
@student id="02" name="Bush" taking="CS2 01"/>
<course id="CS2"/>
```

exercise

What is the problem with the following?

```xml
<!ELEMENT people (person*)>
<!ELEMENT person (name, spouse, children)>
<!ATTLIST person NIN ID #required>
<!ELEMENT spouse (person?) >
<!ELEMENT children (person*)>
```

In an XML document of the DTD, a person is to appear as a child under both his/her father and mother
The limitations of the XML standard (DTD)

- Keys need to 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

Valid XML documents

A valid XML document must have a DTD.
- The document is well-formed
- 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
- Contrast valid documents and well-formed documents (possibly in the absence of a DTD)
DTDs vs. schemas (types)

- By database (or programming language) standard, XML DTDs are rather weak specifications.
  - Only one base type -- PCDATA.
  - No useful “abstractions”, e.g., unordered records.
  - Element type definitions are “global”
  - No methods
  - 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
  - Integrity Constraints

Summary and Review

- DTD provides useful syntactic constraints on documents.
- Element types, attributes
- ID/IDREF, as constraints, are very restricted

Questions:
- How to store large XML documents?
- How to query large documents efficiently?
- How to support XML updates by multiple users simultaneously?
- How to secure access to XML data?
- How to map between XML and other representations?
- How to make XML schemas work like database schemas and programming language types?