Object-Oriented Data to Semi-Structured Data

• Topics
  • OO Implementation challenges
  • Intro to semi-structured data
  • Brief history of XML
  • What is XML?

• Learning objectives
  • Make design tradeoffs in the mechanisms used to implement object-oriented database designs.
  • Explain what semi-structured data is.
  • Explain how XML came to be important.
  • Explain what XML is.
Structured Items

• Objects contain pointers
  • How do you represent pointers on disk?
  • Do you use the same representation in memory?

• Objects contain or are collections
  • How do you represent collections?
  • What if collections are sorted?
  • What if they are arrays which implies a contiguous layout/allocation
  • How do you handle changes in the size of the collection (both growing and shrinking)?
How do you index ADT's?

• If the data type is completely abstract to the database, then you can’t index it, can you?
  • Allow applications to provide indexing techniques (more on this later).
• Even if the database knows about the type, do conventional indexes work?
• Hash tables provide what kind of lookups?
• B+-trees provide what kind of lookups?
• So, what do you do about other kinds of lookup:
  • Image similarity
  • Geospatial proximity (think Google earth)
  • N-dimensional data
  • Sub-spaces of N-dimensional data
  • Query-by-humming
  • Full text search
Extensible Database Systems

• Fundamentally this means letting users add code to your database!
• Movement in the late 80’s/early 90’s
• Pioneered by Postgres
• Commercialized by Miro/Illustra (DataBlades)
• Key idea: let applications download methods, optimizer functions, indexing techniques into your data manager.
• Problems?
Safe Extensions

- Interpreted instead of compiled
- Restricted to safe languages
- Post-processed to check safety
- Implemented as separate process (in separate address space)
- This was the early extensible system work and dovetails nicely with the emergence of safe languages and PCC
Performance Considerations

- Pointers
- Caching
- Concurrency
- Cost estimation and the Optimizer
Pointers

• Would like to use real pointers in memory, but must use OIDs on disk. How do you translate?
• Swizziling: The act of translating from disk addressing to memory addressing
• Disk addressing typically in terms of OID
• Memory addressing in terms of points
• Approaches
  • Swizzle on access (keep table of OID-> memory mappings)
  • Use virtual memory to trigger swizzle
  • Swizzle on eviction
Caching

• All DBMS systems rely on caching to provide reasonable performance.
• What is the unit of caching?
  • Objects
  • Pages
  • Containers (large data segments containing potentially many objects)
• Granularity of caching dictates how much of the database resides on clients versus servers.
• Sometimes also useful to cache query results.
Concurrency

• Probably want granularity for concurrency to equal granularity for caching.
• Concurrency granularity may be critical from a throughput point of view.
Cost Estimation: Optimization

• How does the query processor account for ADTs?
  • Register indexes and relevant statistics
  • Provide callbacks and request user-defined functions
    provide data about distributions, counts, etc.
Semi-Structured Data

• As we move into objects, the details of a schema become fuzzier.
• Yes, objects have specific characteristics, but subclasses suddenly are objects that have different attributes from their superclasses and from each other.
• The semi-structured data model is an attempt to formalize and embrace this reality.
• Characteristics of semi-structured data:
  • Irregular: an item may have missing or additional attributes
  • Parts of the data may lack structure (e.g., images)
  • Some might have a little structure (e.g., text)
  • No fixed schema; evolves to fit data; sometimes just ignored
Semi-Structured Trade-offs

• Advantages
  • Easy to discover new data and load it
  • Easy to integrate heterogeneous data
  • Easy to query without knowing data types

• Disadvantages
  • Loses type information
  • Makes optimization more difficult
Semi-Structured Model

- There are two approaches to semi-structure data:
  - Tree-based (today, XML)
  - Graph-based (next week)
XML: A Brief History

• SGML: Standard Generalised Markup Language
  • ISO Standard in 1986
  • Specification for defining markup languages
  • Derived from GML -- a way of marking up technical documents
• HTML is one particular SGML application
  • A key motivator behind HTML was separating content from presentation.
  • For graphic designers, this is a disaster.
  • HTML got extended to allow presentation specification (e.g., <font>).
  • Then the browser wars ensued.
• And then there was chaos…
And then there was XML

• eXtensible Markup Language
  • Yet another specification for defining markup languages.
  • Then W3C tried to redefine HTML in terms of XML: enter XHTML.
  • Then Microsoft decided to use XML as an interchange format.

• But what is XML???
• Is it a data model?
  • No!
  • It’s a markup language
So why study it in a course on data management?

• It’s use as a markup language makes it interesting from a data management perspective whether it is a data model or not.
• What is a markup language?
  • Way to encode data by embedding information about the data (i.e., embedded meta-data).
  • A way to add computer-readable information to text files.
• XML allows you to create markup languages in which you can define your own (application-specific) tags; tags have no a priori meaning.
• Organization or entire market segments might agree on a set of tags and what they mean. This is called an XML application. (Note: XML applications are not programs.)
So why XML?

- It is a data format.
- It is a data format that is gaining traction.
- There are schema definitions and query syntaxes for XML, so it has some of the features of a data management solution.
- In fact, given our breakdown of data management systems into a top layer consisting of schema and query interfaces and a bottom layer consisting of reliable, robust storage, it fits naturally as an alternative for a top layer.
- Most commercial database companies have embraced XML in some way or another (e.g., XML extensions to relational DBSMs, XPath interfaces, XML to SQL translation, etc).
Overview

• Design Goals
  • Compatible with SGML
  • Easy to parse.
  • Design should be formal and concise

• Basics
  • XML is case sensitive
  • XML names (used universally for tags, attributes, etc)
    • Must start with letter or underscore.
    • May contain any alphanumeric character
    • May contain underscore (_), hyphen (-), and period (.)
    • May not contain any other punctuation
    • No whitespace
    • All names beginning with "XML" are reserved.
Structure (I)

- Elements (tags) are the primary building block.
- Elements are delimited with begin \(<\text{tags}\rangle\) and end \(</\text{tags}\rangle\).
- An XML document is well-formed if:
  - The document starts with an XML declaration:
    \(<\text{?xml version="1.0" encoding="UTF-8" standalone="yes"?>}\>
  - The single root element must contain all the other elements.
  - All elements must be properly nested.
- Attributes
  - Descriptive "parameters" to an element.
  -Specified inside the begin tag.
  - All attributes must be enclosed in quotes.
  - General format:
    \(<\text{tag attr1="val1" attr2="val2"}>\)
Structure (II)

- **Entity References**
  - Shorthand for “special” content.
  - Begin with "&" and end with ";
  - Several entity references are predefined:
    - `&amp;` == &
    - `&apos;` == ’
    - `&lt;` == <
    - `&gt;` == >
    - `&quot;` == "
  - Can use Unicode also: `&#VAL;` where VAL is the unicode value for the desired symbol.
    - `&#x3c;` == < (hexidecimal)
    - `&#60;` == < (decimal)

- **Comments**
  - `<!-- Comments go here -->`
  - Comments can contain anything except --
Structure (III)

- **CDATA**
  - Provides for inclusion of raw text without having to express special characters with entity references (e.g., display XML inside an XML document).
    
    ```xml
    <![CDATA[uninterpreted bytes go here]]>
    ```
  - Can include any characters except the sequence "]]>"
  - The square brackets are part of the syntax.
Example

```xml
<?xml version="1.0"?>
<product dept="dessert">
    <item>Tart, Raspberry</item>
    <inventory>
        <sku>Tartdessq9r0yV</sku>
        <price currency="USD">3.15</price>
        <instock>0</instock>
    </inventory>
    <vendor>Mom’s Kitchen</vendor>
</product>
```
What’s that about no schema?

• There are actually two ways to specify things that look an awful lot like schema:
  • DTD: Document Type Definitions
  • XSD: XML Schema Definition

• DTD
  • Set of rules that specify the legal format of a document.
  • An XML document is valid if it adheres to some DTD.
  • A validating parser compares a document to a DTD and determines if the document adheres to the DTD.
DTD Format (I)

• Begins with a declaration:
  ```xml
  <!DOCTYPE name [DTD declaration]>
  ```
• First thing in a DTD declaration must be the definition of the root element (root is the outermost element in an XML document).
  ```xml
  <!ELEMENT rootname (contentType)>  
  ```
• Five content types:
  • Other elements (specified by a regular expression of the element names)
    • ```xml
      <!ELEMENT memo (to from date body)>  
    ```
  • Parsed character data (#PCDATA)
  • EMPTY (contain no data)
  • ANY (allows any data; reduces how much checking can be done)
  • A regular expression constructed out of the other four types.
DTD Format (II)

- Standard regexp representation
  - \texttt{FOO*} 0 or more FOOS
  - \texttt{FOO+} 1 or more FOOS
  - \texttt{FOO?} 0 or 1 FOO
  - \texttt{FOO, BAR, BAZ}: a FOO followed by a BAR followed by a BAZ

- Attributes
  - Listed outside the element definition.
  - General format:
    - <!ATTLIST elementName (attrName attrType default)+>
  - Parentheses are not part of syntax; part of regexp
Attribute Types

- CDATA: Any string allowed in a well-formed attribute value
- NMTOKEN: name token, like an XML Name except a name token may begin with any legal character (i.e., including numbers, hyphens, and periods).
- NMTOKENS: one or more white-space separated name tokens.
- Enumeration: vertical bar separated list of allowed values
- ID: an XML name that is unique within the document
- IDREF: XML name that refers to an ID type attribute in the document.
- IDREFS: white-space separated list of IDREFs
- ENTITY: name of an unparsed entity declared in the document
- ENTITIES: one or more white-space separated entities
- NOTATION: the name of a notation declared in the document
<!DOCTYPE product [ 
  <!ELEMENT product (item, inventory, vendor)> 
  <!ELEMENT item (#PCDATA)> 
  <!ELEMENT inventory (sku, price, instock)> 
    <!ELEMENT sku (#PCDATA)> 
    <!ELEMENT price (#PCDATA)> 
    <!ATTLIST price currency (USD|NZD|GBP|CAD)> 
    <!ELEMENT instock (#PCDATA)> 
  <!ELEMENT vendor (#PCDATA)> 
    <!ATTLIST product dept (dessert|produce|canned|paper) #REQUIRED> 
]>