ICS 321 Data Storage & Retrieval

Semi-structured Data Model

Asst. Prof. Lipyeow Lim
Information & Computer Science Department
University of Hawaii at Manoa
Schema Variability

- Structured data conforms to rigid schemas.
  - Relational data
- Unstructured data – the other extreme.
  - Eg. Free text
- Certain types of data are inbetween
  - Semi-structured
  - Schema variability across instances as well as time.
  - Eg. E-catalogs
- XML supports a very flexible “schema”

- Model
  - Brand = TOSHIBA
  - Series = REGZA
  - Model = 52HL167
  - Cabinet Color = Black
- Display
  - Screen Size = 52"
  - Recommended Resolution = 1920 x 1080
  - Aspect Ratio = 16:9
  - …

- Model
  - Brand = ViewSonic
  - Model = PJ551D
  - Cabinet Color = Black
  - Type = DLP
- Display
  - Panel = 0.55" DMD
  - Lens = Manual zoom/focus
  - Lamp = 180W, 3,500 hours normal, up to 4,000 eco mode
  - Aspect Ratio = 4:3 (native), 16:9
eXtended Markup Language (XML)

- Design goals:
  - straightforwardly usable over the Internet.
  - support a wide variety of applications.
  - compatible with SGML.
  - easy to write programs which process XML docs.
  - optional features in XML kept to the absolute minimum.
  - human-legible and reasonably clear.
  - easy to create.
  - Terseness in XML markup is of minimal importance.

---

Lipyeow Lim -- University of Hawaii at Manoa
Examples

• Internet:
  – RSS, Atom
  – XHTML
  – Webservice formats: SOAP, WSDL

• File formats:
  – Microsoft Office, Open Office, Apple’s iWork

• Industrial
  – Insurance: ACORD
  – Clinical trials: cdisc
  – Financial: FIX, FpML
  – Mortgages: MISMO

• Many applications use XML as a data format for persistence or for data exchange
XML Document

<dblp>
  <inproceedings key="conf/cikm/HassanzadehKLMW09">
    <author>Oktie Hassanzadeh</author>
    <author>Anastasios Kementsietsidis</author>
    <author>Lipyeow Lim</author>
    <author>Renée J. Miller</author>
    <author>Min Wang</author>
    <title>A framework for semantic link discovery over relational data.</title>
    <pages>1027-1036</pages>
    <year>2009</year>
    <booktitle>CIKM</booktitle>
  </inproceedings>
</dblp>
Processing XML

• Parsing
  – Event-based
    • Simple API for XML (SAX) : programmers write callback functions for parsing events eg. when an opening “<author>” is encountered.
    • The XML tree is never materialized
  – Document Object Model (DOM)
    • The XML tree is materialized in memory

• XML Query Languages
  – XPath : path navigation language
  – XQuery
  – XSLT : transformation language (often used in CSS)
XPath

- Looks like **paths** used in Filesystem directories.
  - Relative vs absolute
- Examples:
  - `/dblp/inproceedings/author`
  - `//author`
  - `//inproceedings[year=2009 and booktitle=CIKM]/title`
- Results are **sequences** of nodes.
- Think of a **node** as the XML fragment for the subtree rooted at that node.
XPath Axes

• An XPath is a sequence of location steps separated by “/” of the form
  – Axisname::nodetest[predicate]

• An axis defines a node-set relative to the current node:
  – self, parent, child, attribute
  – following, following-sibling
  – descendent, descendent-or-self
  – ancestor, ancestor-or-self
  – namespace
  – preceding, preceding-sibling

• Examples
  – /child::dblp/child::inproceedings/attribute::author
    • /dblp/inproceedings/@key
  – /descendent-or-self::author
    • //author
XPath Predicates

• An XPath is a sequence of location steps separated by “/” of the form
  – Axisname::nodetest[predicate]

• Predicates can be comparisons of atomic values or path expressions
  – //inproceedings[ year=“2009” and booktitle=“CIKM”]/title

• A predicate is true if there exists some nodes that satisfy the conditions
  – //inproceedings[author=“Renee ”]
XQuery

- For-Let-Where-Return expressions
- Examples:

```xquery
FOR $auth in doc dblp.xml // author
LET $title = $auth/.. / title
WHERE $author /.. / year = 2009
RETURN
  <author>
    <name>$auth/text()</name>
    <title>$title/text()</title>
  <author>

FOR $auth in doc dblp.xml // author[../ year=2009]
RETURN
  <author>
    <name>$auth/text()</name>
    <title>$auth/.. / title/text()</title>
  <author>
```
XML & RDBMS

• How do we store XML in DBMS?
• Inherent mismatch between relational model and XML data model
• Approach #1: BLOBs
  – Parse on demand
• Approach #2: shredding
  – Decompose XML data to multiple tables
  – Translate XML queries to SQL on those tables
• Approach #3: Native XML store
  – Hybrid storage & query engine
  – Columns of type XML
DB2’s Hybrid Relational-XML Engine

CREATE TABLE Product( id INTEGER, Specs XML );

INSERT INTO Product VALUES(1, XMLParse( DOCUMENT '
  <ProductInfo>
    <Model>
      <Brand>Panasonic</Brand>
      <ModelID>TH-58PH10UK</ModelID>
    </Model>
    <Display>
      <ScreenSize>58in</ScreenSize>
      <AspectRatio>16:9</AspectRatio>
      <Resolution>1366 x 768</Resolution>
    </Display>
  </ProductInfo>
);
SQL/XML

- **XMLParse** – parses an XML document
- **XMLexists** – checks if an XPath expression matches anything
- **XMLTable** – converts XML into one table
- **XMLQuery** – executes XML query

```
SELECT X.*
FROM emp, XMLTABLE ('$d/dept/employee'
    passing doc as "d"
    COLUMNS
    empID INTEGER PATH '@id',
    firstname VARCHAR(20) PATH 'name/first',
    lastname VARCHAR(25) PATH 'name/last')
AS X

SELECT XMLQUERY(  
    '$_doc//item[productName="iPod"]'  
    PASSING PO.Porder as "doc")
AS "Result"
FROM PurchaseOrders PO;
```
XML Storage (DB2 pureXML)

- **String IDs** for Namespace, Tag names
- **Path IDs** for paths
- XML tree partitioned into regions & packed into pages.
- **Regions index** track the pages associated with the XML structure
XML Indexing

• Users create specific **value indexes** associated with specific XPaths.

```sql
CREATE INDEX idx1 ON dept(deptdoc)
  GENERATE KEY USING XMLPATTERN ‘/dept/employee/name’ AS SQL VARCHAR(35)
```

• Index matching requires both the path and the type to match.
  – Queries involving `/dept/employee/name` and explicitly uses `varchar` or `string` for the type associated with the element can exploit the valued index
B+ Trees for XML Indexing

- For XML value indexes we want to map the value associated with an XML pattern to nodes in the XML data tree
- Key part of index entry is the “value”
- Instead of a rid, an index entry stores the (region ID, node ID)

Lipyeow Lim -- University of Hawaii at Manoa