ICS 421 Spring 2010
Non-Relational DBMS

Asst. Prof. Lipyeow Lim
Information & Computer Science Department
University of Hawaii at Manoa
XML Data Model

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.
- **Common Axes**: child, descendant, parent, ancestor, self
- **Examples**:
  - /dblp/inproceedings/author
  - //author
  - //inproceedings[year=2009 and booktitle=CIKM]/title
XQuery

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

```
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>
```

4/22/2010

Lipyeow Lim -- University of Hawaii at Manoa
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>
);

SELECT id FROM Product AS P
WHERE XMLEXISTS('t/ProductInfo/Model/Brand/Panasonic' PASSING BY REF P.Specs AS "t")
**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

**Example SQL Query**

```sql
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
```

```sql
SELECT XMLQUERY(
  '$doc//item[productName="iPod"]'
PASSING PO.Porder as "doc")
AS "Result"
FROM PurchaseOrders PO;
```
### Resource Description Framework (RDF)

<table>
<thead>
<tr>
<th>ID</th>
<th>Author</th>
<th>Title</th>
<th>Publisher</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isbn0-00-651409-X</td>
<td>Id_xyz</td>
<td>The glass palace</td>
<td>Id_qpr</td>
<td>2000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Homepage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id_xyz</td>
<td>Ghosh, Amitav</td>
<td><a href="http://www.amitavghosh.com">http://www.amitavghosh.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Publisher Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id_qpr</td>
<td>Ghosh, Amitav</td>
<td>London</td>
</tr>
</tbody>
</table>
Nodes can be literals.

Nodes can also represent an entity.

Edges represent relationships or properties.
More formally

• An RDF graph consists of a set of RDF triples
• An RDF triple \((s,p,o)\)
  – “s”, “p” are URI-s, ie, resources on the Web;
  – “o” is a URI or a literal
  – “s”, “p”, and “o” stand for “subject”, “property” (aka “predicate”), and “object”
  – here is the complete triple: \(<http://...isbn...6682>, <http://..//original>, <http://...isbn...409X>\)
• RDF is a general model for such triples
• RDF can be serialized to machine readable formats:
  – RDF/XML, Turtle, N3 etc
RDF/XML

<rdf:Description rdf:about="http://.../isbn/2020386682">
  <f:titre xml:lang="fr">Le palais des mirroirs</f:titre>
  <f:original rdf:resource="http://.../isbn/000651409X"/>
</rdf:Description>
Querying RDF using SPARQL

• The fundamental idea: use graph patterns
• the pattern contains unbound symbols
• by binding the symbols, subgraphs of the RDF graph are selected
• if there is such a selection, the query returns bound resources

```
SELECT ?p ?o
WHERE {subject ?p ?o}
```

Where-clause defines graph patterns. \(?p\) and \(?o\) denote “unbound” symbols
Example: SPARQL

```sparql
SELECT ?isbn ?price ?currency  # note: not ?x!
WHERE {?isbn a:price ?x.
    ?x p:currency ?currency.}
```
Linking Open Data

• Goal: “expose” open datasets in RDF
  – Set RDF links among the data items from different datasets
  – Set up, if possible, query endpoints

• Example: DBpedia is a community effort to
  – extract structured (“infobox”) information from Wikipedia
  – provide a query endpoint to the dataset
  – interlink the DBpedia dataset with other datasets on the Web
@prefix dbpedia <http://dbpedia.org/resource/>.
@prefix dbterm <http://dbpedia.org/property/>.

dbpedia:Amsterdam
  dbterm:officialName "Amsterdam" ;
  dbterm:longd "4" ;
  dbterm:longm "53" ;
  dbterm:longs "32" ;
  dbterm:leaderName dbpedia:Job_Cohen ;
  ...
  dbterm:areaTotalKm "219" ;
  ...

dbpedia:ABN_AMRO
  dbterm:location dbpedia:Amsterdam ;
  ...

Amsterdam

<table>
<thead>
<tr>
<th>Country</th>
<th>Netherlands</th>
<th>North Holland</th>
</tr>
</thead>
<tbody>
<tr>
<td>Province</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Municipality</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mayor</td>
<td>Job Cohen<a href="PvdA">1</a></td>
</tr>
<tr>
<td>Alderman</td>
<td></td>
<td>Lodewijk Asscher</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Carolien Gehrels</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tjerk Herrema</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maarten van Poelgeest</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Marijke Vos</td>
</tr>
<tr>
<td></td>
<td>Secretary</td>
<td>Erik Gertzsen</td>
</tr>
<tr>
<td>Area</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>City</td>
<td>219 km² (84.6 sq mi)</td>
</tr>
<tr>
<td></td>
<td>Land</td>
<td>166 km² (64.1 sq mi)</td>
</tr>
<tr>
<td></td>
<td>Water</td>
<td>53 km² (20.6 sq mi)</td>
</tr>
<tr>
<td></td>
<td>Urban</td>
<td>1,009 km² (387.3 sq mi)</td>
</tr>
<tr>
<td></td>
<td>Metro</td>
<td>1,816 km² (700.8 sq mi)</td>
</tr>
<tr>
<td>Elevation</td>
<td></td>
<td>2 m (7 ft)</td>
</tr>
<tr>
<td>Population (1 October 2008)[2][3]</td>
<td></td>
<td>765,265</td>
</tr>
<tr>
<td></td>
<td>City</td>
<td>765,265</td>
</tr>
<tr>
<td></td>
<td>Density</td>
<td>4,458/km² (11,548/sq mi)</td>
</tr>
<tr>
<td></td>
<td>Urban</td>
<td>1,384,422</td>
</tr>
<tr>
<td></td>
<td>Metro</td>
<td>2,158,372</td>
</tr>
<tr>
<td></td>
<td>Demonym</td>
<td>Amsterdammer</td>
</tr>
<tr>
<td>Time Zone</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Summer (DST)</td>
<td>CET (UTC+1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CEST (UTC+2)</td>
</tr>
<tr>
<td>Postcodes</td>
<td></td>
<td>1011 – 1109</td>
</tr>
<tr>
<td>Area code(s)</td>
<td></td>
<td>020</td>
</tr>
</tbody>
</table>

Website: www.amsterdam.nl
Linking the Data

<http://dbpedia.org/resource/Amsterdam>
  owl:sameAs <http://rdf.freebase.com/ns/...> ;
  owl:sameAs <http://sws.geonames.org/2759793> ;
  ...

<http://sws.geonames.org/2759793>
  owl:sameAs <http://dbpedia.org/resource/Amsterdam>
  wgs84_pos:lat "52.3666667" ;
  wgs84_pos:long "4.8833333";
  geo:inCountry <http://www.geonames.org/countries/#NL> ;
  ...

Google’s Bigtable

“Bigtable is a sparse, distributed, persistent multidimensional sorted map”

• It is a type key-value store:
  – Key: (row key, column key, timestamp)
  – Value: uninterpreted array of bytes

• Read & write for data associated with a row key is atomic

• Data ordered by row key and range partition into “tablets”

• Column keys are organized into column families:
  – A column key then is specified using <family:qualifier>

• Timestamp is a 64 bit integer timestamp in microseconds
Example: Webpages using Bigtable

- Row key = reversed string of a webpage’s URL
- Column keys:
  - contents:
  - anchor:cnnsi.com
  - anchor:my.look.ca
- Timestamps: t3, t5, t6, t8, t9
CouchDB

• A distributed document database server
  – Accessible via a RESTful JSON API.
  – Ad-hoc and schema-free
  – robust, incremental replication
  – Query-able and index-able

• A couchDB document is a set of key-value pairs
  – Each document has a unique ID
  – Keys: strings
  – Values: strings, numbers, dates, or even ordered lists and associative maps
Example: couchDB Document

"Subject": "I like Plankton"
"Author": "Rusty"
"PostedDate": "5/23/2006"
"Tags": ["plankton", "baseball", "decisions"]
"Body": "I decided today that I don't like baseball. I like plankton."

- CouchDB enables views to be defined on the documents.
  - Views retain the same document schema
  - Views can be materialized or computed on the fly
  - Views need to be programmed in javascript
Cassandra

• Another distributed, fault tolerant, persistent key-value store
• Hierarchical key-value pairs (like hash/maps in perl/python)
  – Basic unit of data stored in a “column”: (Name, Value, Timestamp)
• A column family is a map of columns: a set of name:column pairs. “Super” column families allow nesting of column families
• A row key is associated with a set of column families and is the unit of atomicity (like bigtable).
• No explicit indexing support – need to think about sort order carefully!
Example: Cassandra

`mccv` → Users

**Stats**

```
<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>emailAddress</td>
<td>&quot;name&quot;:&quot;emailAddress&quot;, &quot;value&quot;:&quot;<a href="mailto:foo@bar.com">foo@bar.com</a>&quot;</td>
</tr>
<tr>
<td>webSite</td>
<td>&quot;name&quot;:&quot;webSite&quot;, &quot;value&quot;:&quot;<a href="http://bar.com">http://bar.com</a>&quot;</td>
</tr>
<tr>
<td>visits</td>
<td>&quot;name&quot;:&quot;visits&quot;, &quot;value&quot;:&quot;243&quot;</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>emailAddress</td>
<td>&quot;name&quot;:&quot;emailAddress&quot;, &quot;value&quot;:&quot;<a href="mailto:user2@bar.com">user2@bar.com</a>&quot;</td>
</tr>
<tr>
<td>twitter</td>
<td>&quot;name&quot;:&quot;twitter&quot;, &quot;value&quot;:&quot;user2&quot;</td>
</tr>
</tbody>
</table>
```