ICS 421 Spring 2010

Non-Relational DBMS

Asst. Prof. Lipyeow Lim

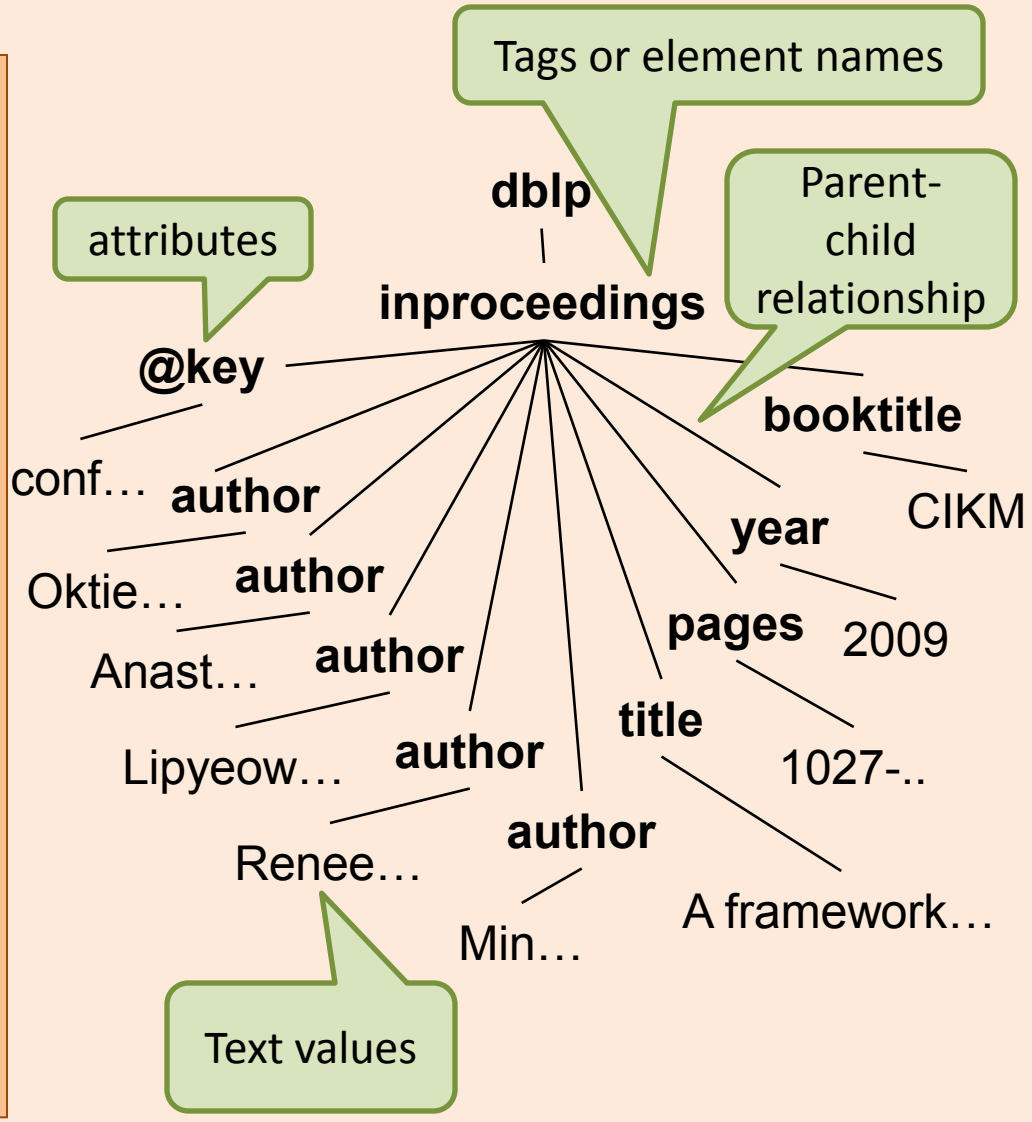
Information & Computer Science Department

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, descendent, 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>
```

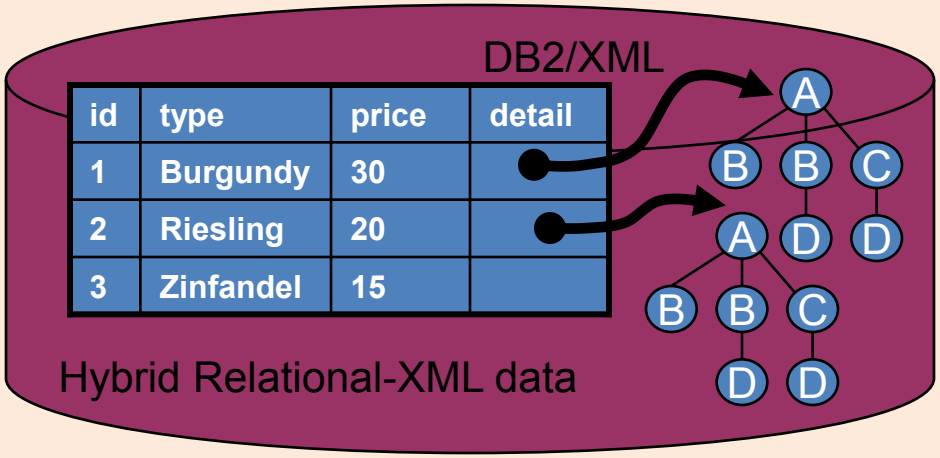
XML & RDBMS

- 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
'<?xml version='1.0'>
      <ProductInfo>
        <Model>
          <Brand>Panasonic</Brand>
          <ModelID>
             TH-58PH10UK
          </ModelID>
        </Model>
        <Display>
          <ScreenSize>58in
          </ScreenSize>
          <AspectRatio>16:9
          </AspectRatio>
          <Resolution>1366 x 768
          </Resolution>
            …
      </ProductInfo>')
   );
```




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

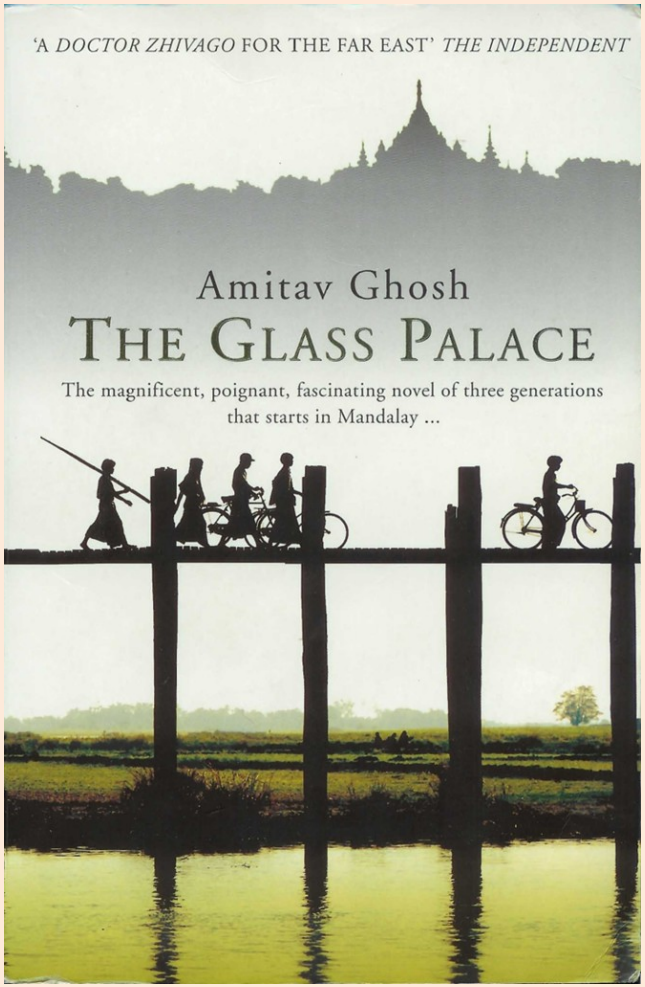
SQL/XML

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

```
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;
```

Resource Description Framework (RDF)

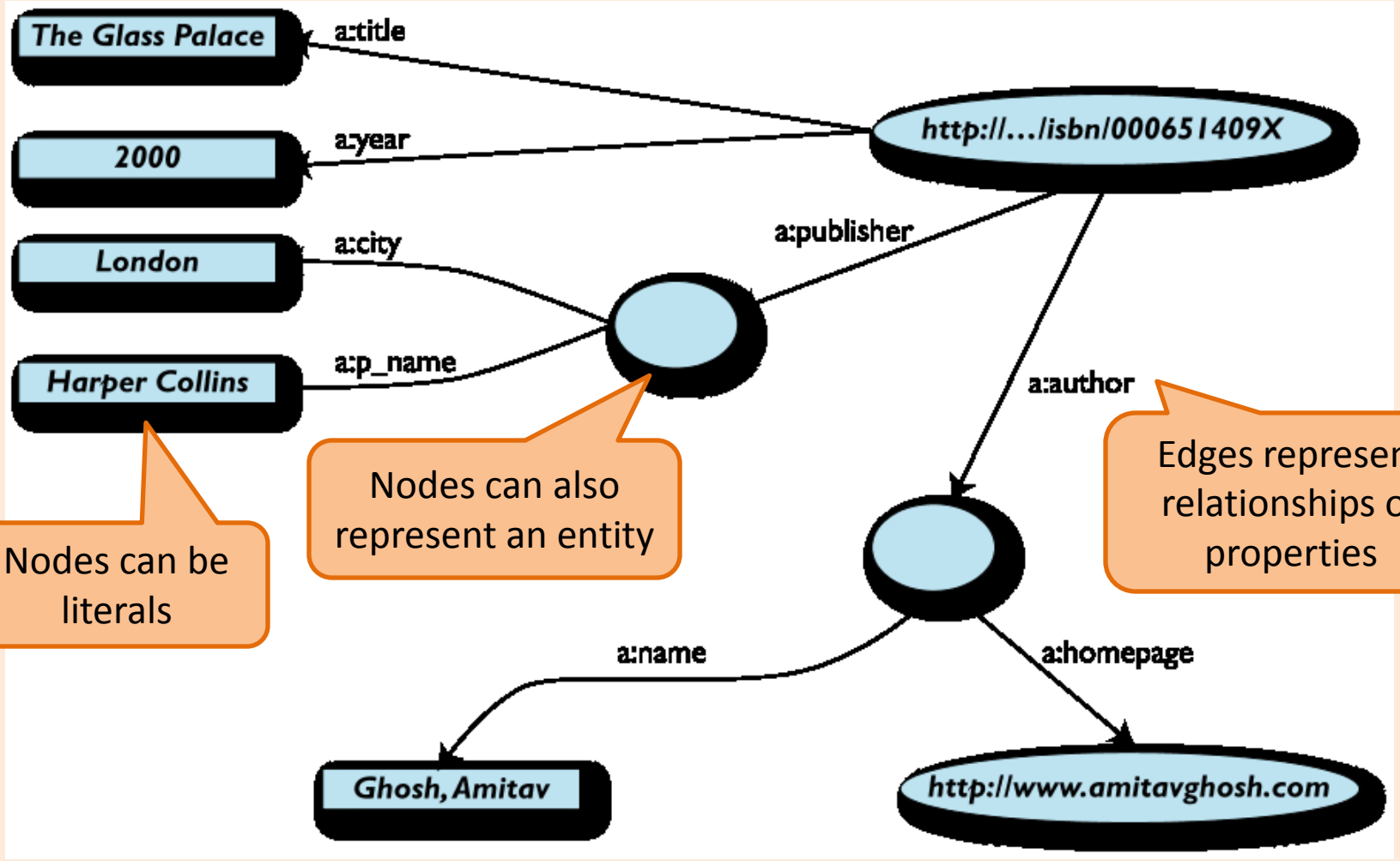


ID	Author	Title	Publisher	Year
Isbn0-00-651409-X	Id_xyz	The glass palace	Id_qpr	2000

ID	Name	Homepage
Id_xyz	Ghosh, Amitav	http://www.amitavghosh.com

ID	Publisher Name	City
Id_qpr	Ghosh, Amitav	London

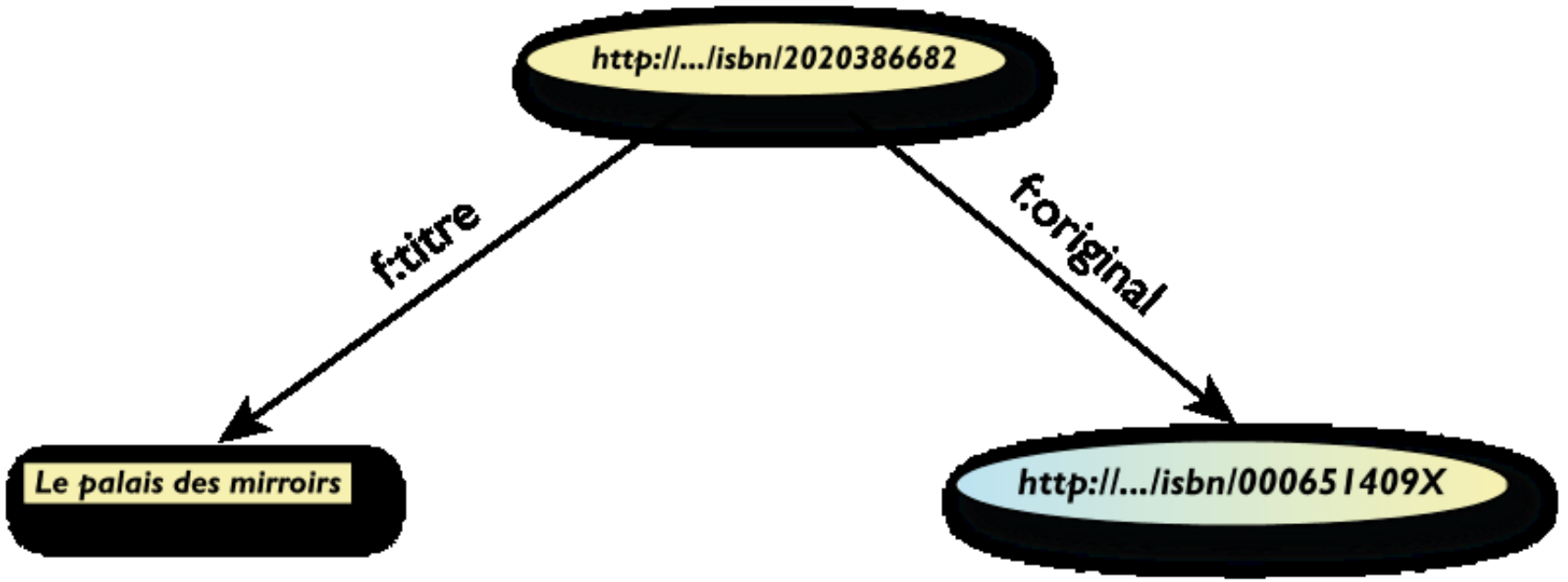
RDF Graph Data Model



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



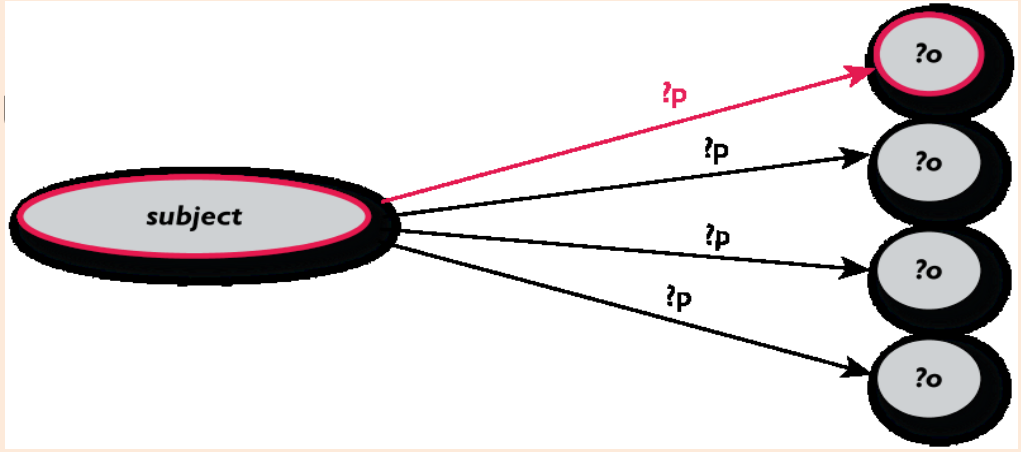

```
<rdf:Description rdf:about="http://…/isbn/2020386682">
    <f:titre xml:lang="fr">Le palais des mirroirs</f:titre>
    <f:original rdf:resource="http://…/isbn/000651409X"/>
</rdf:Description>
```

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 sele
  the query returns bo
  resources

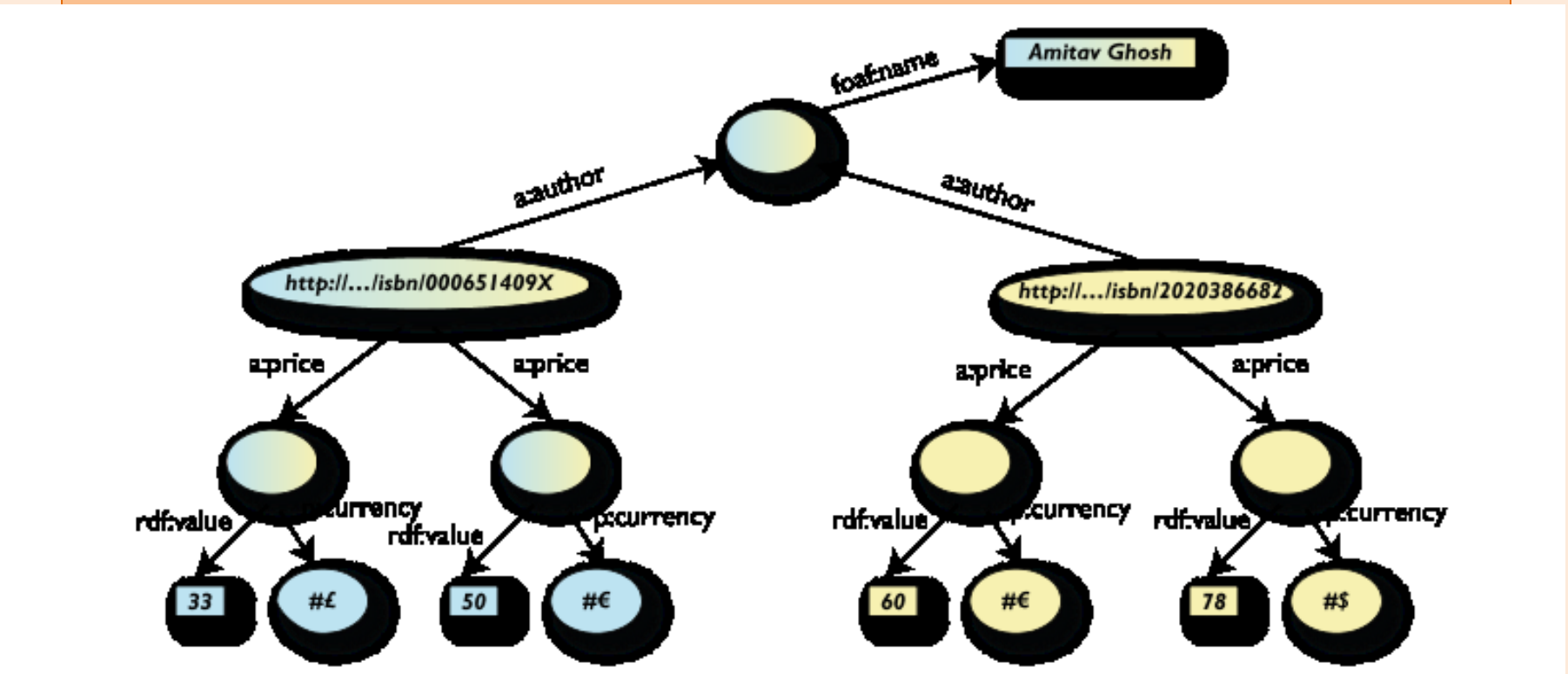
SELECT ?p ?o
WHERE {subject ?p ?o}

Where-clause defines graph patterns. ?p and ?o denote "unbound" symbols



Example: SPARQL

```
SELECT ?isbn ?price ?currency   # note: not ?x!
WHERE {?isbn a:price ?x.
       ?x rdf:value ?price.
       ?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

DBPedia

```
@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

The Keizersgracht at dusk

Location of Amsterdam

Coordinates: 52°22′23″N 4°53′32″E

Country	Netherlands
Province	North Holland
Government	
- Type	Municipality
- Mayor	Job Cohen[1] (PvdA)
- Aldermen	Lodewijk Asscher Carolien Gehrels Tjeerd Herrema Maarten van Poelgeest Marijke Vos
- Secretary	Erik Gerritsen
Area [2][3]	
- City	219 km² (84.6 sq mi)
- Land	166 km² (64.1 sq mi)
- Water	53 km² (20.5 sq mi)
- Urban	1,003 km² (387.3 sq mi)
- Metro	1,815 km² (700.8 sq mi)
Elevation [4]	2 m (7 ft)
Population (1 October 2008)[5][6]	
- City	755,269
- Density	4,459/km² (11,548.8/sq mi)
- Urban	1,364,422
- Metro	2,158,372
- Demonym	Amsterdammer
Time zone	CET (UTC+1)
- Summer (DST)	CEST (UTC+2)
Postcodes	1011 – 1109
Area code(s)	020

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> ;
  ...
```

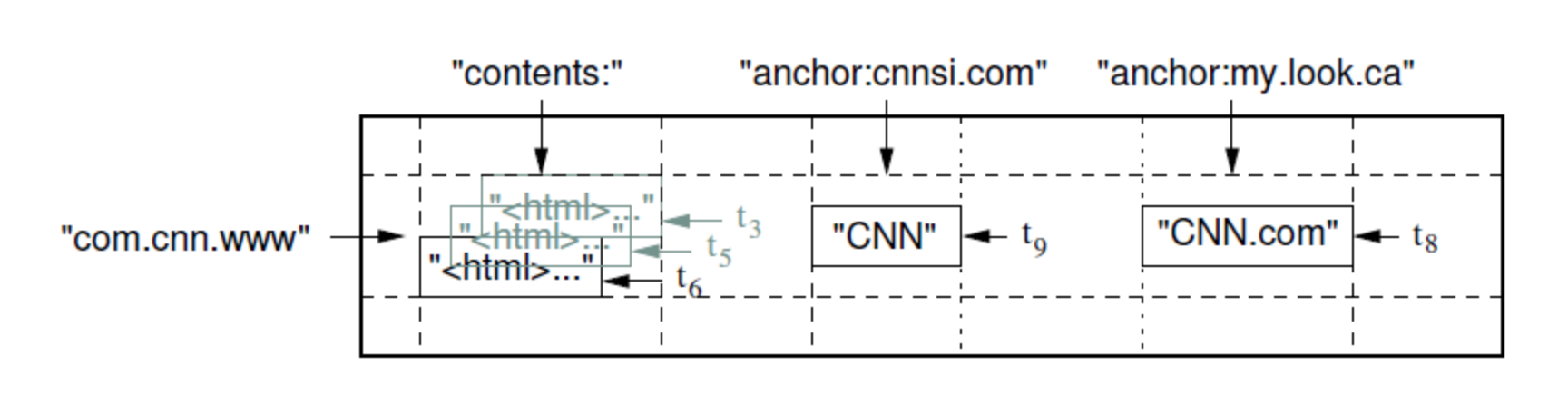
```
<http://sws.geonames.org/2759793>
  owl:sameAs <http://dbpedia.org/resource/Amsterdam>
  wgs84_pos:lat "52.3666667" ;
  wgs84_pos:long "4.8833333";
  geo:inCountry <http://www.geonames.org/countries/#NL>
;
  ...
```

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