XML Data Management – An Overview

Swetha Machanavajhala

Database Seminar

Structured Data

0.00	$2.07	$2.00	$1.50	$0.00	$0.90	8.97	5720.79	6.47	16.52	14.21	30.73	33.69	$1,010.80
2.50	$3.77	$2.50	$1.00	$1.40	$0.00	15.17	5735.96	11.2	17.00	14.02	31.02	32.55	$976.43
1.50	$9.07	$3.50	$0.00	$0.70	$0.30	33.07	5768.73	15.1	17.65	14.09	31.74	33.15	$994.37
9.00	$10.47	$0.50	$1.50	$1.40	$0.00	46.87	5815.60	22.9	18.14	14.38	32.52	34.08	$1,022.43
2.50	$4.44	$2.00	$1.50	$3.15	$0.00	33.84	5849.44	13.6	17.49	14.39	31.88	33.58	$1,007.52
5.50	$13.72	$1.50	$0.50	$4.20	$0.90	44.57	5893.11	26.3	17.76	14.34	32.10	33.45	$1,003.55
2.00	$4.72	$2.50	$0.00	$1.05	$0.00	37.27	5930.38	10.3	16.40	15.13	31.52	33.09	$992.62
2.00	$1.56	$0.00	$0.00	$0.35	$0.60	10.51	5940.29	4.51	16.16	15.23	31.39	31.51	$945.38
1.00	$4.00	$0.00	$1.00	$1.05	$0.00	14.05	5954.34	7.05	16.08	15.27	31.35	29.91	$897.27
3.50	$13.42	$7.50	$0.50	$0.70	$0.30	48.07	6002.11	25.9	16.85	15.71	32.56	31.06	$931.67
8.00	$8.23	$4.00	$2.50	$1.05	$0.00	42.68	6044.79	23.8	17.44	15.65	33.09	32.04	$961.09
4.00	$7.90	$2.00	$1.25	$4.20	$0.30	43.95	6088.44	19.7	17.12	16.17	33.28	32.28	$968.37
3.00	$6.78	$2.50	$2.50	$1.05	$0.00	44.63	6133.07	15.8	16.78	16.94	33.72	32.34	$970.29
2.00	$6.20	$1.50	$1.50	$0.70	$0.00	36.50	6169.57	11.9	15.31	17.55	32.87	31.79	$953.70
1.00	$1.26	$0.00	$0.25	$0.35	$0.00	10.96	6180.53	2.86	15.06	17.95	33.01	30.95	$928.44
0.00	$3.83	$1.50	$2.00	$1.40	$0.90	18.63	6198.26	9.63	14.95	18.31	33.26	30.91	$927.25
8.50	$9.74	$3.00	$0.00	$1.40	$0.00	42.44	6240.70	22.6	15.49	18.44	33.93	31.78	$953.46
3.50	$9.17	$2.00	$1.50	$1.40	$1.20	41.87	6281.37	18.8	15.19	18.38	33.57	32.81	$984.22
3.00	$9.73	$5.50	$0.50	$5.25	$0.00	49.78	6331.15	24	15.94	18.77	34.71	33.70	$1,011.02
4.00	$11.00	$3.00	$2.00	$21.00	$0.00	72.20	6403.35	41	16.99	19.70	36.68	34.91	$1,047.20
1.00	$7.12	$1.00	$1.50	$3.50	$0.00	40.52	6443.87	14.1	17.26	19.65	36.91	34.83	$1,044.93

Spreadsheets

- Data resides in fixed fields within a record or file.
- Has a fixed schema.
- Contains information stored in columns and rows.
- Has an identifiable structure understood by computers.
- Well organized for human readers.

PubID	Publisher	PubAddress
03-4472822	Random House	123 4th Street, New York
04-7733903	Wiley and Sons	45 Lincoln Blvd, Chicago
03-4859223	O'Reilly Press	77 Boston Ave, Cambridge
03-3920886	City Lights Books	99 Market, San Francisco

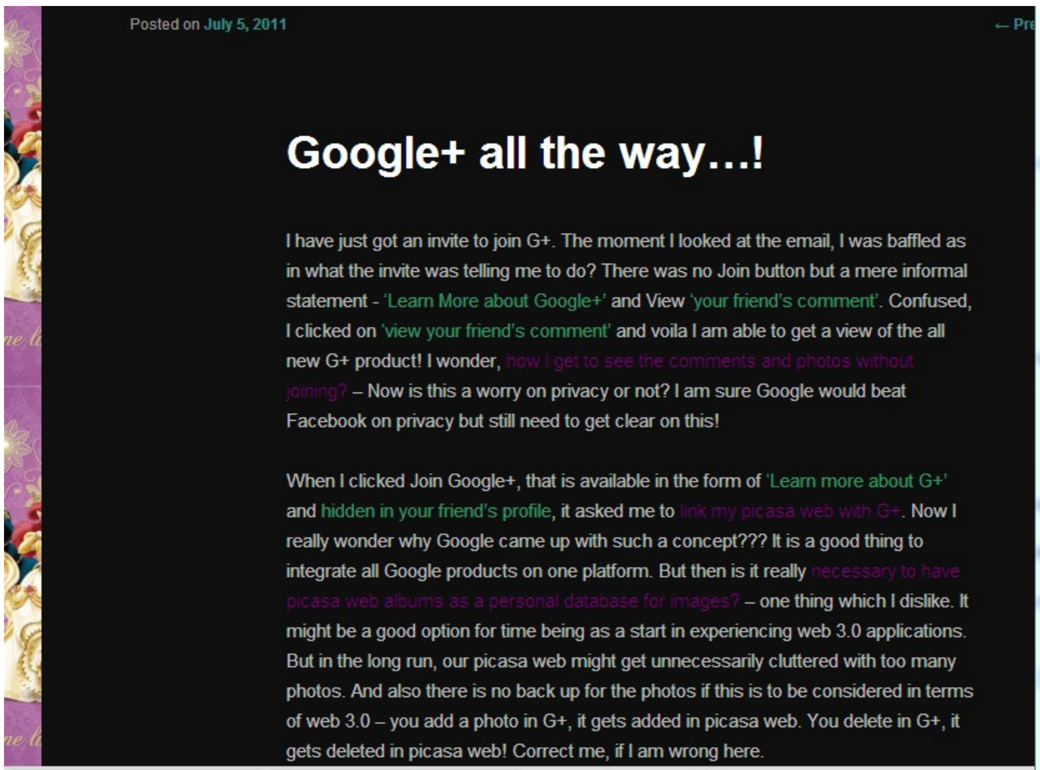
AuthorID	AuthorName	AuthorBDay
345-28-2938	Haile Selassie	14-Aug-92
392-48-9965	Joe Blow	14-Mar-15
454-22-4012	Sally Hemmings	12-Sept-70
663-59-1254	Hannah Arendt	12-Mar-06

ISBN	AuthorID	PubID	Date	Title
1-34532-482-1	345-28-2938	03-4472822	1990	Cold Fusion for Dummies
1-38482-995-1	392-48-9965	04-7733903	1985	Macrame and Straw Tying
2-35921-499-4	454-22-4012	03-4859223	1952	Fluid Dynamics of Aquaducts
1-38278-293-4	663-59-1254	03-3920886	1967	Beads, Baskets & Revolution

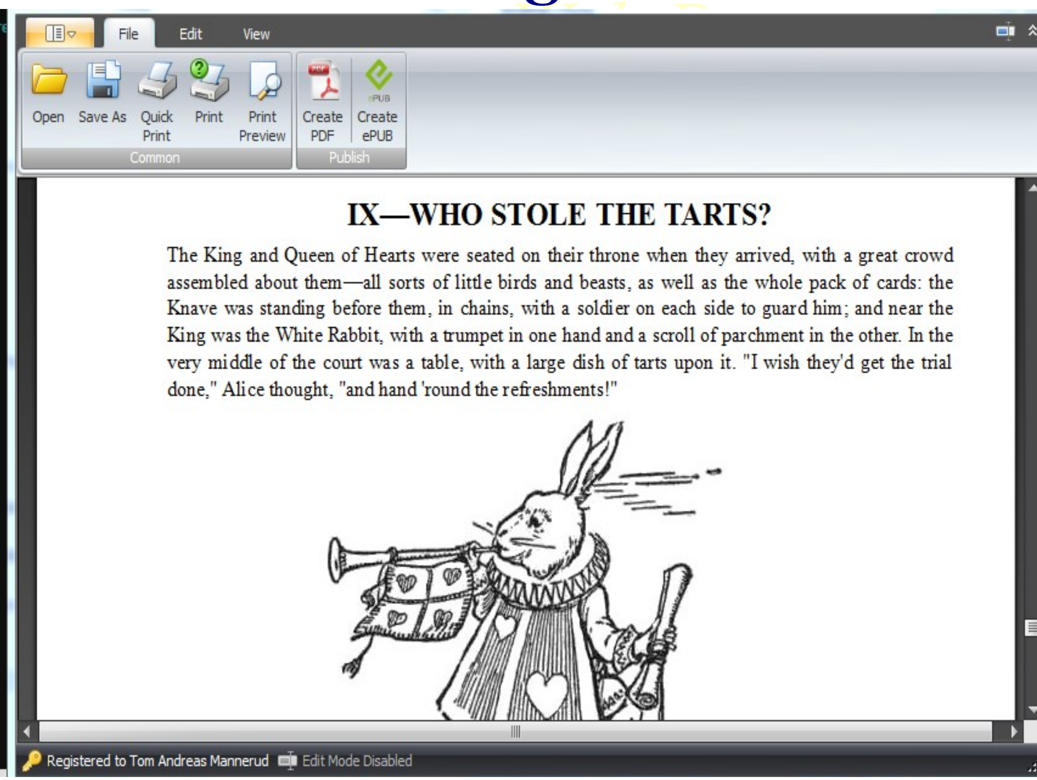
Relational Databases

Unstructured Data

Blogs



Word Processing Documents



- Currently most of the data are unstructured.
- Data has minimal structure like “text” in <Title> vs text in <Body>
- Does not fit well into relational tables.

Why XML?

Current data is in the form of Web Documents.

Data from different sources contain different schema. Cannot model this data using RDBMS.

- XML is known for its flexible schema

Need to structure this data such that it can be fit into a RDBMS.

Need to handle, store, query and exchange data across different systems and architectures.

- Semi structured / Unstructured data consists of data objects whose attributes are not known in advance.

- XML contains self-describing tags that can structure these data objects.

- These tags describe "what" data represent – Useful for sharing data between applications.

- Not easy to query such data using SQL. So we go for pure XML databases.

Example of an XML Document

<Presenters> ← **[Root Element node]**

<Presenter @name = “Swetha”> ← **[Attribute Node]**

<topic> XML Data Management </topic>

</Presenter>

<Presenter> ← **[Descendants of “Presenter” element → <Paper> , <topic> and <name>]**

<paper>

<topic> Map Reduce </topic> ← **[Element node]**

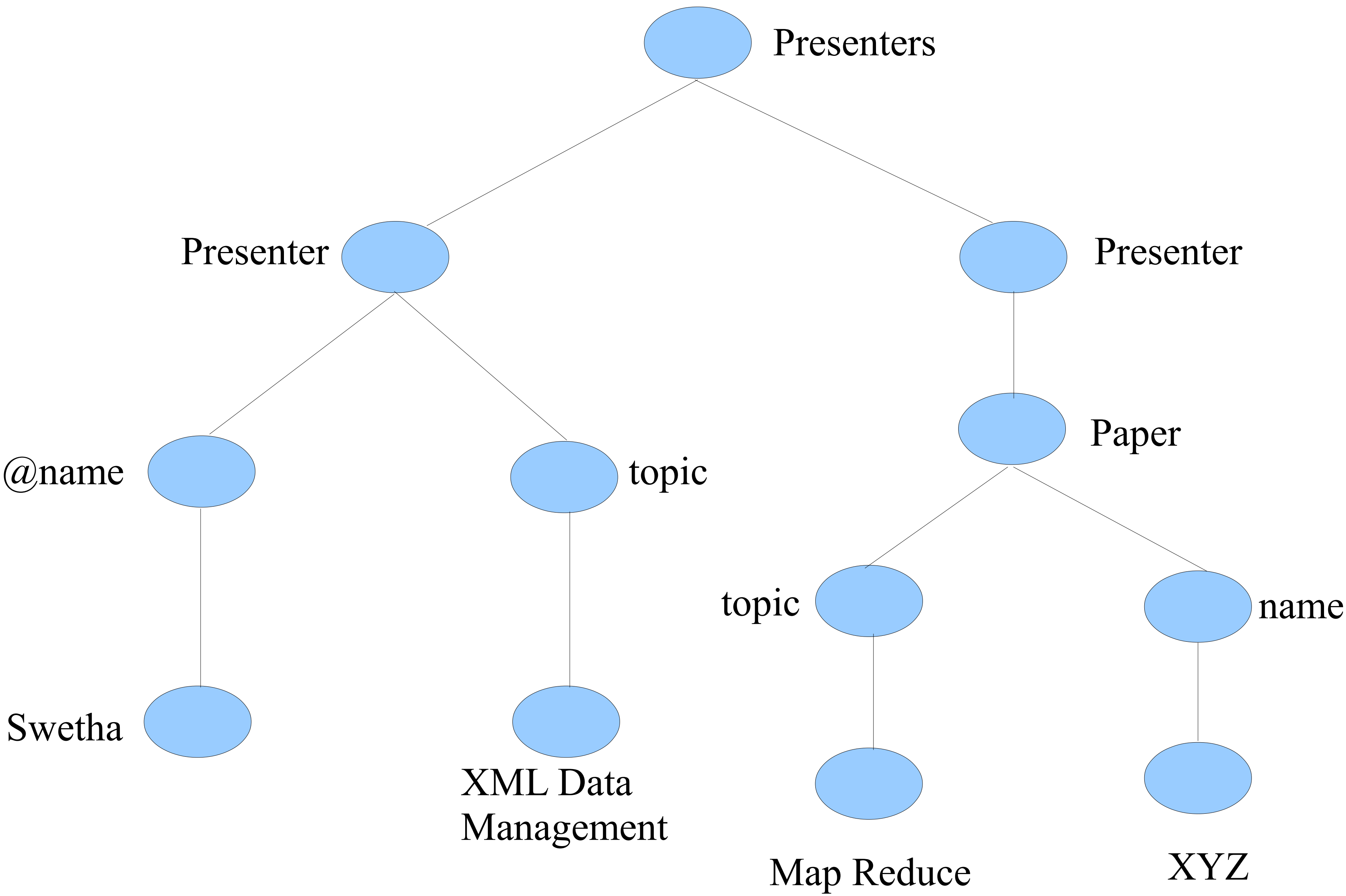
<name> XYZ </name> ← **[Ancestors of “name” element → <Paper> , <Presenter> and <Presenters>]**

</paper>

</Presenter>

</Presenters>

Basic Model: Tree



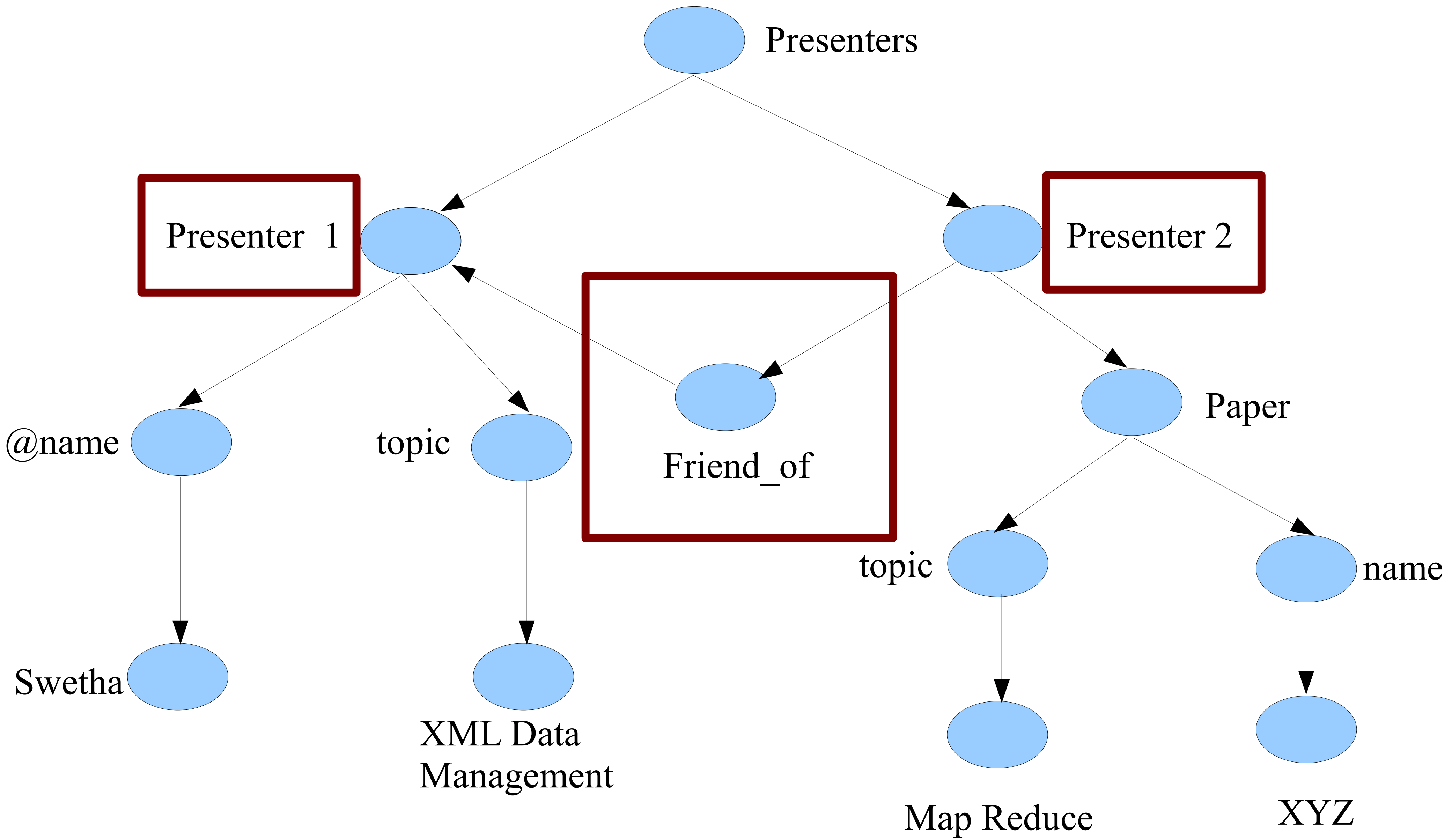
Representing Primary and Foreign Keys

ID attribute uniquely identifies an element

IDREF attribute refers to other elements identified by ID attributes.

```
<Presenters>
    <Presenter ID = “1”>
       <paper>
          <topic> XML Data Management </topic>
          <name> Swetha </name>
       </paper>
    </Presenter>
    <Presenter ID = “2” Friend_Of IDREF = “1”>
       <paper>
          <topic> Map Reduce </topic>
          <name> XYZ </name>
       </paper>
    </Presenter>
</Presenters>
```

Extended Model: Directed Acyclic Graphs



XML Queries – Relational Approach

1. XPath

Based on structural hierarchical navigation through elements and attributes in an XML document.

Selecting Nodes:

2 commonly used axes:

'/' - Child axis → "A/B"

Select all B-tagged child nodes of A-tagged nodes.

'//' - Descendant axis → "A//B"

Select all B-tagged descendant nodes of A-tagged nodes

XPath – An Example

//Presenter/topics → returns all topics under the element node of "Presenter"

→ **Path Pattern**

/Presenter[@name = Swetha]/topic → returns the topic of presenter named Swetha.

[Predicate]

XPath query with a predicate represents a **"Twig Pattern"** → Returns exactly one output node!

XML Queries – Relational Approach (contd.)

2. Xquery

- Xquery for XML same as SQL for databases.
- Designed to query XML files and databases that appear as XML.

Composed Of:

***F**or-**L**et-**W**here-**R**eturn (FLWR) clauses.*

Usage:

- Search Web documents for relevant details.
- Extract information to use in a web service.
- Transform XML data to XHTML

XQuery – An Example

Select the topics of presenter named Swetha

We have the following path expression:

//Presenters/Presenter[@name = Swetha]/topic

FLWR equivalent of the above expression:

```
For $x in //Presenters/Presenter/topic
Where $x/name = “Swetha”
Return $x/topic
```

XML Queries – IR Style Approach

Information Retrieval – Style XML queries are used to query ***text-dense*** XML documents.

Text-dense

Value elements in XML document involve long text.

In the previous examples, value elements are not text-dense.

Why IR-Style approach?

Need to ***search large texts*** that total in the order of billions to trillions of words.

Allows ***Ranked Retrieval*** → return the best answer to the query among many documents.

Database-style approach using Xpath and Xquery does not support the above.

Boolean IR Queries

Scenario: A collection of Shakespeare's Plays. Determine which play of Shakespeare contain the words Brutus AND Caesar NOT Calpurnia.

Linear scan through the text → not a good option for large texts.

We need to index the documents in advance.

Done using Binary term – document ***Incidence Matrix*** where Terms are the indexed units.

Plays in columns

Words in rows

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	…
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
…							

"Brutus" appears in Play "Antony and Cleopatra"

Boolean IR Queries (contd.)

Solution to the query Brutus AND Caesar AND NOT Calpurnia

Consider the vectors for each of the terms.

110100 AND 110111 AND NOT(010000)

→ 110100 AND 110111 AND 101111 = 100100

Look up the incidence matrix for the result

Result → Antony and Cleopatra and Hamlet.

DB + IR Queries

Enhances database-style XML queries like Xpath and Xquery with IR-style characteristics.

Example, add “Contains” function to Xpath query as we have seen previously:

/Presenter[contains (“Databases” , “Swetha”]/ @name

Returns names of all presenters whose (child or descendant) subelements contain approximate matches to keywords “Databases” and “Swetha”

Storing & Querying XML Data efficiently...

Approach 1: Relational Approach

Leverage RDBMS by mapping XML to Relational Tables.

Approach 2: Native Approach

Perform navigation, insertion, deletion and update operations using optimized operators on a tree-structured data model.

1. XML Query Processing: Relational Approach

Main Idea:

- Shred XML documents into relational tables.
- Transform XML queries to SQL queries for querying the database.

How is this done?

- There are many approaches but we will look into 2 basic approaches.

  - Basic Edge Approach
  - Binary Approach

Basic Edge Approach

Key Idea:

- ***Assign an ID*** to every node of an XML tree.
- Store information about an edge in a row in ***Edge Table***
- Edge Table representation:

Edge_Table(Source_ID , Ordinal Number , Target_ID , Label, Flag , Value)

[Source node in the XML Tree]

[Order of outgoing edge from Source]

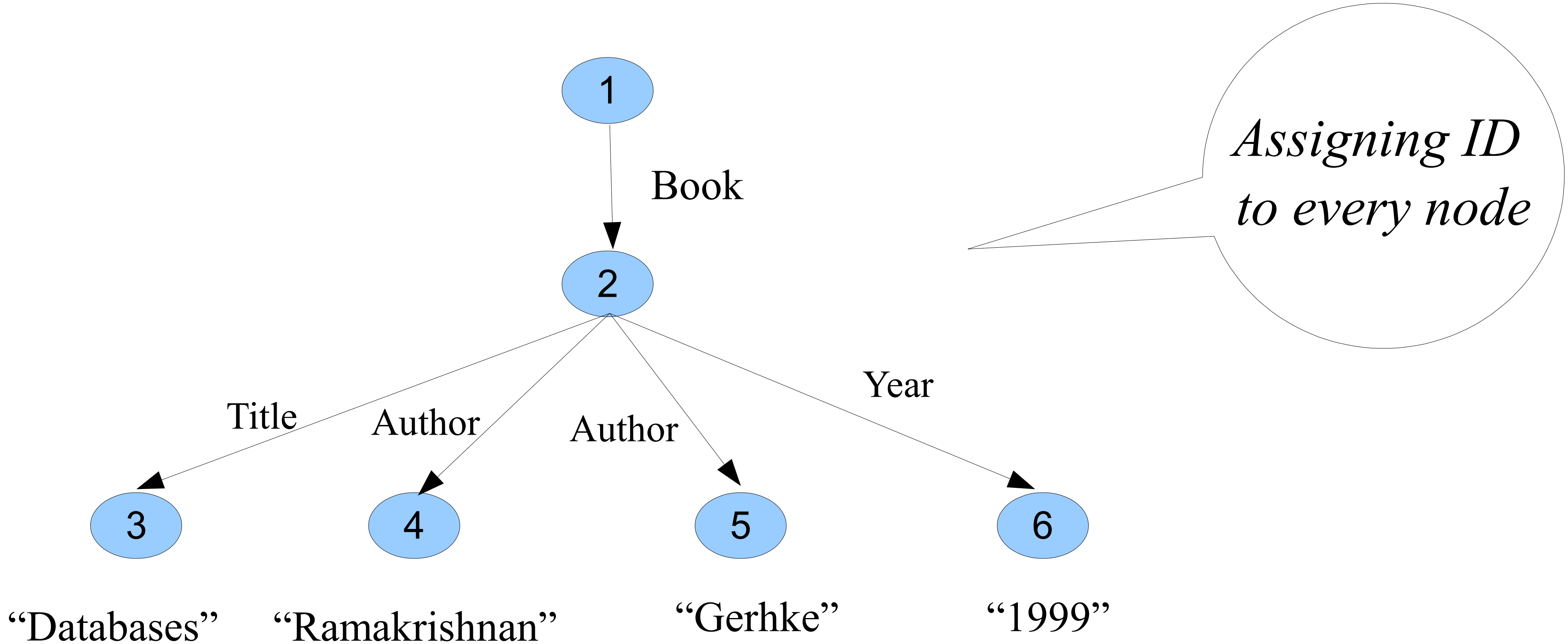
[Target node to which the current node is pointing]

[Tag on the edge]

[Type of target node]

[Value of target node]

An Example… Step 1



Step 2: Edge Table

Source ID	Ordinal Number	Target ID	Label	Flag	Value
1	1	2	Book	Ref	-
2	1	3	Title	Val	Databases
2	2	4	Author	Val	Ramakrishnan
2	3	5	Author	Val	Gerkhe
2	4	6	Year	Val	1999

Step 3: Transform XML query to SQL

SQL Query for "/Book[title = "Databases"]/year"

Select year, Value

From Edge Book, Edge title, Edge year

Where Book.label = 'book' and

title.label = 'title' and

year.label = 'year' and

book.Source = 1 and

book.Target = title.source and

book.Target = year.source and

title.Value = 'Databases'

→ ***[Edge Selection]*** (lines: Book.label, title.label, year.label)

→ ***[Edge Joining]*** (lines: book.Source, book.Target = title.source, book.Target = year.source)

Efficiency of Basic Edge Approach

- **Helps in shredding XML data into relations.**

- **Can query the tables using SQL.**

- **However retrieving data for each edge in edge selection part can lead to slow processing.**

- **Need to speed up the processing of this section.**

Binary Approach

- Pregroups all edges in Edge table by their labels and creates one table for each distinct label.
- Each label has the following schema:

  Label(Source, Target, Flag, Value)

- Example:

  Table 1: Book (1 , 2 , Ref , -)

  Table 2: Title (2 , 3 , Val , Databases) ...

SQL Query using Binary Approach

→ SQL Query for "/Book[title = "Databases"]/year"

Select year, Value

From Book , title , year

Where book.Source = 1 and
book.Target = title.source and
book.Target = year.source and
title.Value = 'Databases'

Avoiding the edge selection part speeds up processing!

Trade-off: Creating multiple tables for each label in large XML documents can be chaotic!

2. XML Query Processing – Native Approach

Why Native approach?

- Relational approach does not exhibit optimal query processing performance.
- Storage and query processing tailored for XML data only.

How is data stored?

- Inverted Lists!
- Create an inverted list for each distinct tag in the XML document.

How is the location of an element defined?

- Represented as (Start, End, Level) numbers.

Inverted List

XML Document

```
  1
<Presenters>
      2
    <Presenter>
          3      4          5
        <name> Swetha </name>
          6     7   8        9            10
        <topic> XML Data Management </topic>
     11
    </Presenter>
      12
    <Presenter>
          13
        <paper>
              14    15  16        17
            <topic> Map Reduce </topic>
              18     19     20
            <name> XYZ </name>
          21
        </paper>
     22
    </Presenter>
 23
</Presenters>
```

Inverted List

Each distinct tag is stored in an inverted list.

Syntax:
(Start , End , Level) numbers.

<Presenter> (2,11,1) , (12,22,1)

<name> (3,5,2) , (18,20,3)

The Multi-Predicate MerGe JoiN (MPMGJN) Approach

- Useful for querying "A//B" or "A/B"

  Procedure:

  Initialize 2 cursors to point to 2 inverted lists.

- Consider <Presenter> list as ListA(start , end)

  <name> list as ListB(start , end)

- Positions within the lists are compared at each iteration

- Presenter (2 , 11) ; Name (3, 5)

- a.start = 2 , a.end = 11 ; b.start = 3 , b.end = 5

Algorithm…

```
If  cursor_B.start < cursor_A.start  Then
   advance cursor_B;
Else
   temp_cursor_B = cursor_B;
   While( temp_cursor_B.start < cursor_A.end )   // the inner-loop join
      Output a tuple solution into join results. Specifically,
           Case 1 (For the `A/B' query):
               Output (cursor_A, temp_cursor_B) if cursor_A.level+1 = temp_cursor_B.level;
           Case 2 (For the `A//B' query):
               Output (cursor_A, temp_cursor_B);
      advance temp_cursor_B;
   Endwhile
   advance cursor_A;
```

Native Approach - Efficiency

- Experimental results showed that MPMGJN approach is faster than current RDBMS join implementations.

- Each element in list B is iterated   to find which B's are children of A for executing query A/B. This leads to more processing time.

- Processing time can be reduced by adopting other native methods such as Stack based approach.

Open Issues

- Can RDBMS be efficiently leveraged to query XML data ?
- Would a combined approach of relational databases and native methods be better?
- How to process queries for large XML data?

Conclusion...

What did we see?

- Need for XML
- How to map XML to Relational Tables.
- Opt for IR-Style queries in case of large texts.
- Efficiently processing XML queries.
- Relational approach – transform XML to SQL queries.
- Native approach – query the data stored in special data structures like inverted lists.
- Open Issues.

XML is not a replacement for HTML but an extension to it!!!

Reads that might interest you...

[1] http://vgc.poly.edu/~juliana/pub/xml-data-management-slides.pdf

[2] http://plato.asu.edu/slides/yi.pdf

[3] How to Store and Query XML Data, *Silvia Stefanova*

[4] Efficiently Querying Large XML Data Repositories: A Survey, *Gang Gou and Rada Chirkova*

Thank You...