Relational Approach

(COSC 416)

Nazli Goharian
nazli@cs.georgetown.edu

Slides are mostly based on Information Retrieval Algorithms and Heuristics, Grossman, Frieder


Problem Definition

- Three conceptual data types:
  - Structured:
    - Objective (absolute) correctness; perfect accuracy; efficiency only issue (e.g.; relational database)
  - Unstructured:
    - Subjective (relative) correctness; accuracy & efficiency trade-off (e.g.; documents, images, video, sound).
  - Semi-structured:
    - Combination of structured and unstructured (e.g.; XML)

- Problem:
  - Until recently, state of the art individually queried each type specific system and merged answers.

Since the early 1990’s our IR Lab and its predecessors’ work has been to seamlessly integrate the searching of multiple data types.
IR as application of RDBMS

- Typical IR functionality may be accommodated:
  - Boolean query
  - Relevance Ranking – multiple similarity measures
  - Proximity Search

- Advantages are:
  - Integration
  - “Free” parallel processing of IR
  - Leverages work done to optimize DBMS
  - Portability

Relational Inverted Index

All inverted index entries

\[
\begin{array}{c|c}
\text{term} & \text{docID} \\
\hline
\text{vehicle} & D1 \\
\text{vehicle} & D3 \\
\text{vehicle} & D4 \\
\end{array}
\]

e.g., vehicle \rightarrow D1, D3, D4 results in:
The Nikkei Stock Average closed at 29,754.73 points up 156.92 points on the Tokyo Stock Exchange Wednesday.

Simplistic Models:

Keyword and Boolean Searches

Relational Approach:

*Keyword Search*

- Keyword search

  ```
  select i.docID
  from INDEX i, QUERY q
  where i.term = q.term
  ```

- Keyword search with stop word list

  ```
  select i.docID
  from INDEX i, QUERY q, STOPLIST s
  where (i.term = q.term) and (i.term <> s.term)
  ```
Relational Approach:

**Boolean Search OR query**

\[
\begin{align*}
\text{select } & \text{docID} \\
\text{from } & \text{INDEX} \\
\text{where } & \text{term} = \text{term1} \\
\text{union} & \\
\text{select } & \text{docID} \\
\text{from } & \text{INDEX} \\
\text{where } & \text{term} = \text{term2} \\
\text{union} & \\
\text{select } & \text{docID} \\
\text{from } & \text{INDEX} \\
\text{where } & \text{term} = \text{term3} \\
\text{union} & \\
\vdots & \\
\text{select } & \text{docID} \\
\text{from } & \text{INDEX} \\
\text{where } & \text{term} = \text{termN}
\end{align*}
\]

Relational Approach:

**Boolean Search AND query**

\[
\begin{align*}
\text{select } & \text{docID} \\
\text{from } & \text{INDEX} \\
\text{where } & \text{term} = \text{term1} \\
\text{intersect} & \\
\text{select } & \text{docID} \\
\text{from } & \text{INDEX} \\
\text{where } & \text{term} = \text{term2} \\
\text{intersect} & \\
\text{select } & \text{docID} \\
\text{from } & \text{INDEX} \\
\text{where } & \text{term} = \text{term3} \\
\text{intersect} & \\
\vdots & \\
\text{select } & \text{docID} \\
\text{from } & \text{INDEX} \\
\text{where } & \text{term} = \text{termN}
\end{align*}
\]
Fixed Join-Count AND Queries

Find all documents that contain all of the terms found in the QUERY relation:

\[
\text{select } i.\text{docID} \\
\text{from INDEX } i, \text{ QUERY } q \\
\text{where } i.\text{term} = q.\text{term} \\
\text{group by } i.\text{docID} \\
\text{having } \text{count (distinct (i.term))} = \\
\text{select count (distinct (term)) from QUERY}
\]

TAND Queries

Find all documents that contain at least \( X \) of the terms found in the QUERY relation:

\[
\text{select } i.\text{docID} \\
\text{from INDEX } i, \text{ QUERY } q \\
\text{where } i.\text{term} = q.\text{term} \\
\text{group by } i.\text{docID} \\
\text{having } \text{count (distinct (i.term))} \geq X
\]
Relevance Ranking:

Vector Space
and
Probabilistic Models

Relational Document Representation
(Single Term Processing)

<table>
<thead>
<tr>
<th>DOCID</th>
<th>docname</th>
<th>headline</th>
<th>dateline</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>AP81214-0028</td>
<td>Stocks Up In Tokyo</td>
<td>TOKYO (AP)</td>
</tr>
</tbody>
</table>

INDEX

<table>
<thead>
<tr>
<th>docID</th>
<th>termcnt</th>
<th>term</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>1</td>
<td>nikkei</td>
</tr>
<tr>
<td>28</td>
<td>2</td>
<td>stock</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>average</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>closed</td>
</tr>
<tr>
<td>28</td>
<td>2</td>
<td>points</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>up</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>tokyo</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>exchange</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>wednesday</td>
</tr>
</tbody>
</table>

TERM

<table>
<thead>
<tr>
<th>term</th>
<th>idf</th>
</tr>
</thead>
<tbody>
<tr>
<td>average</td>
<td>1.08</td>
</tr>
<tr>
<td>closed</td>
<td>1.08</td>
</tr>
<tr>
<td>exchange</td>
<td>1.00</td>
</tr>
<tr>
<td>nikkei</td>
<td>2.07</td>
</tr>
<tr>
<td>points</td>
<td>1.23</td>
</tr>
<tr>
<td>stock</td>
<td>1.00</td>
</tr>
<tr>
<td>tokyo</td>
<td>1.58</td>
</tr>
<tr>
<td>up</td>
<td>0.30</td>
</tr>
<tr>
<td>wednesday</td>
<td>0.60</td>
</tr>
</tbody>
</table>
Relational Query Representation

**ORIGINAL QUERY:**
“nikkei stock exchange
american stock exchange”

<table>
<thead>
<tr>
<th>Term</th>
<th>tf</th>
</tr>
</thead>
<tbody>
<tr>
<td>nikkei</td>
<td>1</td>
</tr>
<tr>
<td>Stock</td>
<td>2</td>
</tr>
<tr>
<td>exchange</td>
<td>2</td>
</tr>
<tr>
<td>american</td>
<td>1</td>
</tr>
</tbody>
</table>

**Term Frequency** (tf): number of occurrences of term $t_k$ in document $i$

**Document Frequency** (df): number of documents which contain $t_j$

**Inverse Document Frequency** (idf): $\log(d/df_j)$ where $d$ is the total number of documents

**Notes:**
- $idf$ is a measure of uniqueness of a term **across the collection**
- $tf$ is the frequency of a term **in a given document**
Similarity Coefficients

- Several similarity coefficients based on the query vector $X$ and the document vector $Y$ are defined:

### Inner Product

$$ \sum_{i=1}^{t} x_i \cdot y_i $$

### Cosine Coefficient

$$ \frac{\sum_{i=1}^{t} x_i y_i}{\sqrt{\sum_{i=1}^{t} x_i^2 \cdot \sum_{i=1}^{t} y_i^2}} $$

Relevance Ranking:

**SQL for VSM dot product**

*List all documents in the order of their similarity coefficient where the coefficient is computed using the dot product*

$$(\text{Query Weight} \cdot \text{Document Weight})$$

**SQL:**

```
SELECT d.docID, d.docname, SUM(q.termcnt * t.idf * i.termcnt * t.idf)
FROM QUERY q, INDEX i, TERM t, DOCUMENT d
WHERE q.term = i.term AND
  q.term = t.term AND
  i.docID = d.docID
GROUP BY d.docID, docname
ORDER BY 3 DESC
```
Relevance Ranking:
SQL for Probabilistic Retrieval

\[
\sum_{t \in \text{term}} \log \left( \frac{(\text{numdocs} - \text{df}_t) + 5}{(\text{df}_t + 5)} \right) \cdot \left( \frac{2.2 \cdot \text{tf}_{i,t}}{0.3 + (0.75 \cdot \text{doclength} / \text{avgdoclength}) + \text{tf}_{i,t}} \right) \cdot q_{tf}
\]

SELECT d.docID, d.docname, SUM(
    LOG(((\text{NumDocs} - \text{t.df}) + 0.5) / (\text{t.df} + 0.5)) *
    ((2.2*\text{i.tf}) / (0.3 + ((0.75 * d.DocLen)/\text{AvgDocLen}) + \text{i.tf})) * q\text{.termcnt}
) FROM INDEX i, TERM t, DOCUMENT d, QUERY q
WHERE i.term = t.term
AND i.docID = d.docID
AND t.term = q.term
GROUP BY d.docID, d.docname
ORDER BY 3;

Relational Document Representation
(Phrase Processing)

<table>
<thead>
<tr>
<th>docID</th>
<th>docname</th>
<th>headline</th>
<th>dateline</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>AP881214-0028</td>
<td>Stocks Up In Tokyo</td>
<td>TOKYO (AP)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>docID</th>
<th>termcnt</th>
<th>phrase</th>
<th>idf</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>1</td>
<td>nikkei stock</td>
<td>2.49</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>stock average</td>
<td>3.33</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>average closed</td>
<td>2.14</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>points up</td>
<td>2.61</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>tokyo stock</td>
<td>2.14</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>stock exchange</td>
<td>1.34</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>exchange wednesday</td>
<td>2.10</td>
</tr>
</tbody>
</table>

Simple Phrase Parsing

- Simple phrase parser with the following rules
  - Phrases do not include stop terms
  - Phrases do not span across symbols

*Example:* The Nikkei Stock Average closed at 29,754.73 points up 156.92 points, on the Tokyo Stock Exchange Wednesday.

**Phrases:**
- nikkei stock
- stock average
- average closed
- points up
- tokyo stock
- stock exchange
- exchange wednesday

Caveat: Logical Design versus Physical Implementation

While the design shown represents the replication of the document identifier, in the physical implementation, clustered tables are actually used.

That is, attribute values that are logically repeated many times are physically clustered by the attribute value to eliminate the replication storing only one copy for each unique attribute value. (Note clustered tables in Oracle 8)

The I/O to retrieve a posting list is achieved via a grouped block read as opposed to retrieval across distributed storage.

Clustered Tables:
Posting List Processing

<table>
<thead>
<tr>
<th>term</th>
<th>docid</th>
<th>tf</th>
</tr>
</thead>
<tbody>
<tr>
<td>stock</td>
<td>( 1, 1 ),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( 28, 2 ),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( 111, 5 ),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( 1024, 3 ),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( 2263, 11 ),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( 11101, 12 ),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( 12345, 1),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( 13456, 6),</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( 23456, 4)</td>
<td></td>
</tr>
</tbody>
</table>
Clustering Tables: Relevance Feedback Processing

**TRADITIONAL**

<table>
<thead>
<tr>
<th>docID</th>
<th>termcnt</th>
<th>phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>1</td>
<td>nikkei</td>
</tr>
<tr>
<td>28</td>
<td>2</td>
<td>stock</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>average</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>closed</td>
</tr>
<tr>
<td>28</td>
<td>2</td>
<td>points</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>up</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>tokyo</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>exchange</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>wednesday</td>
</tr>
</tbody>
</table>

**CLUSTERED**

<table>
<thead>
<tr>
<th>docID</th>
<th>termcnt</th>
<th>phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>{ ( 1</td>
<td>nikkei ),</td>
</tr>
<tr>
<td></td>
<td>( 2</td>
<td>stock ),</td>
</tr>
<tr>
<td></td>
<td>( 1</td>
<td>average ),</td>
</tr>
<tr>
<td></td>
<td>( 1</td>
<td>closed ),</td>
</tr>
<tr>
<td></td>
<td>( 2</td>
<td>points ),</td>
</tr>
<tr>
<td></td>
<td>( 1</td>
<td>up ),</td>
</tr>
<tr>
<td></td>
<td>( 1</td>
<td>tokyo ),</td>
</tr>
<tr>
<td></td>
<td>( 1</td>
<td>exchange ),</td>
</tr>
<tr>
<td></td>
<td>( 1</td>
<td>wednesday ) }</td>
</tr>
</tbody>
</table>

Relational XML Approach: Architecture

- **Semistructured Query**
- **SQL Query**
- **DB**
- **XML Collection**
- **Database Results**

XML Search

• XML provides “tags” that allow both structured and unstructured data to be represented in the same XML document.

• Frequently used as a common representation for a variety of document formats.

Introduction

• XML: Extensible Markup Language
• Defined by the WWW Consortium (W3C)
• Derived from SGML (Standard Generalized Markup Language), but simpler to use than SGML
• Documents have tags giving extra information about sections of the document
  – E.g. `<title> XML </title> <slide> Introduction … </slide>
• Extensible, unlike HTML
  – Users can add new tags, and separately specify how the tag should be handled for display

• A wide variety of tools is available for parsing, browsing and querying XML documents/data
XML Introduction (Cont.)

• Tags make data (relatively) self-documenting
  – E.g.
    
    ```xml
    <university>
      <department>
        <dept_name> Comp. Sci. </dept_name>
        <building> Taylor </building>
        <budget> 100000 </budget>
      </department>
      <course>
        <course_id> CS-101 </course_id>
        <title> Intro. to Computer Science </title>
        <dept_name> Comp. Sci </dept_name>
        <credits> 4 </credits>
      </course>
    </university>
    ```

Structure of XML Data

• **Tag**: label for a section of data
• **Element**: section of data beginning with `<tagname>` and ending with matching `</tagname>`
• Elements must be properly nested
  – Proper nesting
    • `<course> ... <title> ... </title> </course>`
  – Improper nesting
    • `<course> ... <title> ... </course> </title>`
  – Formally: every start tag must have a unique matching end tag, that is in the context of the same parent element.
• Every document must have a single top-level element
Example of Nested Elements

```xml
<purchase_order>
  <identifier> P-101 </identifier>
  <purchaser> .... </purchaser>
  <itemlist>
    <item>
      <identifier> RS1 </identifier>
      <description> Atom powered rocket sled </description>
      <quantity> 2 </quantity>
      <price> 199.95 </price>
    </item>
    <item>
      <identifier> SG2 </identifier>
      <description> Superb glue </description>
      <quantity> 1 </quantity>
      <unit-of-measure> liter </unit-of-measure>
      <price> 29.95 </price>
    </item>
  </itemlist>
</purchase_order>
```

Tree Model of XML Data

- An XML document is modeled as a tree, with nodes corresponding to elements and attributes
  - Element nodes have child nodes, which can be attributes or subelements
  - Text in an element is modeled as a text node child of the element
  - Children of a node are ordered according to their order in the XML document
  - Element and attribute nodes (except for the root node) have a single parent, which is an element node
  - The root node has a single child, which is the root element of the document
Relational XML Approach: Storage

<book>
  <author>
    John Smith
  </author>
  <title>
    Colonial America
  </title>
  <publisher state="NY">
    NY Publishing
  </publisher>
</book>

<table>
<thead>
<tr>
<th>pinndx</th>
<th>pinndxnum</th>
<th>parent</th>
<th>tagtype</th>
<th>tagpath</th>
<th>atname</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>E</td>
<td>1</td>
<td>1</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>E</td>
<td>2</td>
<td>1</td>
<td>John Smith</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>E</td>
<td>3</td>
<td>1</td>
<td>Colonial America</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>E</td>
<td>4</td>
<td>1</td>
<td>NY Publishing</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>A</td>
<td>4</td>
<td>2</td>
<td>NY</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tagpath</th>
<th>value</th>
<th>atname</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[book]</td>
<td>1</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>[book,author]</td>
<td>2</td>
<td>state</td>
<td></td>
</tr>
<tr>
<td>[book,title]</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[book,publisher]</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

XQuery

- XQuery is a general purpose query language for XML data
- Currently being standardized by the World Wide Web Consortium (W3C)
  - The textbook description is based on a January 2005 draft of the standard. The final version may differ, but major features likely to stay unchanged.
- XQuery is derived from the Quilt query language, which itself borrows from SQL, XQL and XML-QL
- XQuery uses a
  
  ```
  for ... let ... where ... order by ... result ...
  ```

  syntax
  
  `for` ⇔ SQL `from`
  `where` ⇔ SQL `where`
  `order by` ⇔ SQL `order by`
  `result` ⇔ SQL `select`
  `let` allows temporary variables, and has no equivalent in SQL

Relational XML Approach: Translation

```xml
let $item := /order/item
where $item[customer_id=3 and credit_card/type="VISA"]
order by $item/id
Return
  <item>
    <id>{$item/id}</id>
    <price>{$item/price}</price>
  </item>
```

```sql
SELECT DISTINCT q3.value, q3.pinndxnum, q4.value, q4.pinndxnum
FROM pinndx q0, pinndx q1, pinndx q2, pinndx q2_1, pinndx q3, pinndx q4
WHERE q0.tagpath=1 AND q1.nvalue=3 AND q1.tagpath=4 AND q1.parent=q0.pinndxnum AND q2.tagpath=5 AND q2.value="VISA" AND q2.parent=q2_1.pinndxnum AND q2_1.parent=q0.pinndxnum AND q3.tagpath=7 AND q3.parent=q0.pinndxnum AND q4.tagpath=8 AND q4.parent=q0.pinndxnum ORDER BY q4.nvalue
```