Data Models and Information Accesses –
CSV888-Special Module
Lecture 2
2015

(set, graph, map, archetype)
(relations, XML, KML, ADL)
(list)

-Subhash Bhalla
Application Design and Development

- Application Programs and User Interfaces
- Tree Structured Data
  (a) Client-side - XHTML, CSS, JavaScript
  (b) Storage side - RDBMS, XML, JSON, YAML, DOT, ...
- Web Fundamentals - Client + web server + Database Server
- Servlets and JSP, PHP
- Application Architectures
- Rapid Application Development
- Application Performance
- Application Security
- Encryption and Its Applications
Programs and Data

• Program $\leftrightarrow$ Data
  $\rightarrow$ Direct access to the data/medium
  $\rightarrow$ (format - csv, space, Columns - data types, ...variables (hardwired to data)

  Program [ structured data - JAVA/C++ objects]

Database - 3 level Data Dictionary
  - Structured Data (DBMS (scheme): db (data))
    - sets, relations, list, bags

Web Data - Semi-structured Data (latex, HTML, ...)
  - Objects, object-class, sets, ....
Data Interchange

- Program 1 → CSV (comma Separated values)
- Program 2 ← CSV values

- Program data dump
- Stacks
- ORACLE Database Dump
- Arrays
- Abstract Data Types

- PROGRAMMING → to upload and process
- → knowledge of syntax and semantics

- NEW TRENDS (data sharing among multiple applications)
  → YAML, Jason, XML, Candle
Information Interchange

- Information System 1 → Amazon Java books
- Information Systems 2 ← Amazon Books

1. Objects - Books, Rooms with id and (x,y) coordinates, Students, Courses, ....
2. Documents - web documents, financial statements of companies, ...
3. Graphs and structures - Proteins, Maps, ...
- Information → Sets (RDB), relation ! DB !
- → Tree-structured Data (XML), ...

→ Syntax and Semantics
- Tree Structured Data → XML, JSON, Candle Markup
Old Data Models - List Processing

1. Hierarchical Model - Tree (rooted, acyclic, unique path from root to leaf)

2. Network Model - Linked list

1) and 2) → influenced by list structure
Basic Data Elements

• 1. Set - No duplicates and no order
   [ (3,1,1) - not a set; Set (3,1) is same as set (1,3) ]

• 2. Bag - data has no order
   [(3,1,1) is same as (1,3,1)]

• 3. List - has order [(3,1,1) is not same as (1,3,1)]
Content has no form - an island...

1. \( \rightarrow \text{Set} \)

Set = Relation;

2. Stored over List

3. List \( \rightarrow \) Processed by Von Neumann architecture / Turing M/C
Abstract Data Type (ADT)

- Abstract data type (ADT) → a mathematical model for a certain class of data structures that have similar behavior;

→ for certain data types of one or more programming languages that have similar semantics.

→ An abstract data type is defined indirectly (by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations)

- Example, an abstract stack → defined by three operations: push, pop, and peek

→ When analyzing the efficiency of algorithms that use stacks, one may also specify that all operations take the same time no matter how many items have been pushed into the stack, and that the stack uses a constant amount of storage for each element.
**Abstract data types** are purely theoretical entities,

1. → used to simplify the description of abstract algorithms,
2. → to classify and evaluate data structures,
3. → to formally describe the type systems of programming languages.
4. → ADT may be implemented by specific data types or data structures, in many ways and in many programming languages;
5. → or described in a formal specification language.
6. → ADTs are often implemented as modules: the module's interface declares procedures that correspond to the ADT operations, sometimes with comments that describe the constraints.
7. → This information hiding strategy allows the implementation of the module to be changed without disturbing the client programs.
Content: Table - Set/bag (represent as?)

<table>
<thead>
<tr>
<th>company</th>
<th>section</th>
<th>employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>s1</td>
<td>e1</td>
</tr>
<tr>
<td>c1</td>
<td>s1</td>
<td>e2</td>
</tr>
<tr>
<td>c1</td>
<td>s2</td>
<td>e3</td>
</tr>
</tbody>
</table>

• <company id="c1">  
  • <section id="s1">  
    • <employee id="e1"/>  
    • <employee id="e2"/>  
  • </section>  
  • <section id="s2">  
    • <employee id="e3"/>  
  • </section>  
• </company>
Data in Tree (list)

```
<employeeList>
  <employee id="e1">
    <company id="c1"/>
    <section id="s1"/>
  </employee>
  <employee id="e2">
    <company id="c1"/>
    <section id="s1"/>
  </employee>
  <employee id="e3">
    <company id="c1"/>
    <section id="s2"/>
  </employee>
</employeeList>
```
Relational Model (set)- EF Codd 1971 (IBM)

• A. Two levels-
  1. User → Sets and set operations

---------------------------------------

2. Storage → list ;
  • -User [need elements]; (no navigation)
    -Storage [store over list; provide thru index or list search]
  • User [need set operations]
    ← do them on your own
### Table form $\rightarrow$ set (product set)

<table>
<thead>
<tr>
<th>company</th>
<th>section</th>
<th>employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>s1</td>
<td>e1</td>
</tr>
<tr>
<td>c1</td>
<td>s1</td>
<td>e2</td>
</tr>
<tr>
<td>c1</td>
<td>s2</td>
<td>e3</td>
</tr>
</tbody>
</table>

- **Table form of data** $\rightarrow$ set, or bag
- **Operations** $\rightarrow$ set operations
  - Query language
Comparison of methods

• Old Models- Hierarchical Model
  → variation over list structure
  → Started from Bottom: Query on list

Network Model → variation over list

Relational Model: Top Down Approach →
Set + Set operations : Two layers
→ No navigation as in old models
→ influence over query operations
Text book - research book

• 1. SQL
  a) 1971- 1976
  b) SQL 2 ( 1992 )
  c) SQL 3 ( Object - Relational Data Models ) (1999)
[ New ] DB Forms of content

1. Web Documents

2. Map → Google Map, Yahoo map, MS map

3. Bio-medical informatics → web data resources (complex chains of molecules in proteins)

4. Electronic Health Records
New DB Forms of content

- Content $\rightarrow$ has a form (structure)
  (not islands of data)
- Representation $\rightarrow$ 1. list (too simple)
  2. set
  3. graph

Low level (Disk/Memory) $\leftarrow$ list
Processing $\rightarrow$ Content; intermediate
representation (may be); storage (list)
[ New ] DB Forms of Content

• Web Document $\rightarrow$ XML
• Web-based Maps $\rightarrow$ KML (google)
• Bio-Medical Data Resources $\rightarrow$ XML, or similar to XML
• Electronic Health Records $\rightarrow$ ADL (similarities with XML, used in conjunction with XML)

• 1. Document form $\rightarrow$ graph (not set)
A graph $G = (V,E)$ is a collection of nodes (vertices) and edges.

A graph “relationship structure” among different data elements.

A graph database is a collection of different graphs representing different relationship structures.

Notes:
   a) Storage Level → list structures, b) multiple levels, c) intermediate forms (XML → Lists)
Compare: Graph database and (set) Relational database

A relational database maintains different instances of the same relationship structure (represented by its ER schema)

A graph database maintains different relationship structures

← Web Documents, maps, Bio-Medical Informatics, Electronic Health Records

← Store in intermediate forms- XML, KML, ADL
Queries over New DB Contents

• Attribute Queries (Type A)
  - Queries over attributes and values in nodes and edges. (Equivalent to a relational query within a given schema)

• Structural Queries (type B)
  [Not Main focus of our Discussion]
  - Queries over the relationship structure itself. Examples: Structural similarity, substructure, template matching, etc.
Graph Database Applications - (Type A and Type B)

- **Software Engineering**
  - UML diagrams, flowcharts, state machines, ...
- **Knowledge Management**
  - Ontologies, Semantic nets, ...
- **Bioinformatics**
  - Molecular structures, bio-pathways, ...
- **CAD**
  - Electrical circuits, IC designs, ...
- **Cartography, XML Bases, HTML Webs, **
Structural Queries on Graph Data (Type B)

- Undirected Graphs
  - Structural similarity, substructure
- Directed Graphs
  - Structural similarity, substructure, reachability
- Weighted Graphs
  - Shortest paths, “best” matching substructure
- Labeled Graphs
  - Labeled structural similarity, unlabeled structural similarity
Structural Queries (Type B)

- **Substructure query**
  - Given a graph database \( G = \{G_1, G_2, ..., G_n\} \) and a query graph \( Q \), return all graphs \( G_i \) where \( Q \) is a subgraph of \( G_i \).

- **Structural similarity**
  - Given a graph database \( G = \{G_1, G_2, ..., G_n\} \) and a query graph \( Q \) and a threshold \( t \), return all graphs \( G_i \) where the *edit distance* between \( Q \) and \( G_i \) is at most \( t \).
  - The edit distance between two graphs is the number of edge modifications (additions, deletions) required to rewrite one graph into the other.
Data $\rightarrow$ Graph

- Storage Models for Graphs
- Data Models for Graph Databases
- Structural Indexes
- Mining Frequent Subgraphs
  - gSpan (graph-based Substructure pattern mining)
  - FBT (Graph Data and Mining)
Structural Queries

• In graph databases structure matching has to be performed against a set of graphs!

• Method of storage, pre-processing and index structures → crucial
  (if structural searches are to be practical)
Storing Graph Data → set

Attributed Relational Graphs (ARGs)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>B</td>
<td>q</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>C</td>
<td>s</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>D</td>
<td>t</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>C</td>
<td>p</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>D</td>
<td>r</td>
</tr>
</tbody>
</table>
Storing Graph Data

• ARGs
  - ARGs store a graph as a set of rows, each depicting an edge
  - Amenable to storage in an RDBMS and easy attribute searches using SQL
  - New Query Languages (↔Research Type A)
  - Costly structural searches, requiring complex nesting of SELECT statements
  - Each graph needs a separate table
  - Type B (VLDB, SIGMOD, many forums)
1. Storing Graph Data in XML (rooted tree, acyclic, unique path from root)

- `<node id="A">
-   <node id="B">
-     <node id="C">
-       <node id="D">
-         <node id="C">
-           <node id="D">
-             <node id="A">
-             </node>
-           </node>
-         </node>
-       </node>
-     </node>
-   </node>
- </node>`
2. Storing Graph Data in XML
(arbitrary graph)

XML with IDs and IDREFS:

```xml
<node id="A", adj="C D">
  <node id="B">
    <node id="C">
    </node>
    <node id="D">
    </node>
  </node>
</node>
```

![Graph Diagram]

A → B → C → D
Storing Graph Data

- **XML (with or without IDREFS)**
  - Reduces graph database to an XML base
  - Use XPath / XQuery engines for attribute queries and structural queries
  - Widely supported by a variety of XML parsers
  - Costly structure/sub-structure matching
  - Needs distinction between IDREF edges and hierarchy edges
1. Input ISBNs or Keywords (of author or title).

2. Send request data to Amazon Web Service.


4. Extract Documents from the response.

5. Add update data and state data to the book catalog.

6. Store these data into KB.
KB: Data Structure

Book

- URL of image: text
- ASIN: number (※1)
- Title: text
- Average rating: number (※2)
- Author name: text
- URL of detail page: text
- Price: text
- Publisher name: text
- Publication date: text
- Number of pages: text
- Sales rank: number
- ........
Web Documents

• Web Services: Web API
  1. Amazon E-Commerce Service
  2. Yahoo! Search Web Service
  3. Google AJAX Search API
  4. Technorati Search API

• XML DB: DBMS for XML
  1. Knowledge Base (KB)
     • A collection of book data for BUS.
  2. Information Repository of User’s Needs (IRUN)
     • A collection of data that consists of user’s interest and needs.
Amazon E-Commerce Service

- Product information (e.g. catalogs, reviews, rating) retrieval for:
  1. Books
  2. Music
  3. DVD
  4. Electronics
  5. Kitchen
  6. Software
  7. Video Games
  8. Toys
Yahoo! Search Web Service

• Web information (e.g. URL, content or hit count) retrieval:

1. Web pages
2. Images
3. Movies
Google AJAX Search API

- Embed search box in a web page and display search results of:

  1. Web pages
  2. News
  3. Video
  4. Maps
  5. Blogs
Book Utilization System

Web Service Handler
- Google AJAX Search API
- Technorati Search API
- Yahoo! Search Web Service
- Amazon E-Commerce Service

Web User Interface
- Blogs
  - Display
  - Retrieval
- Alternate Keywords
  - Search & Suggestion
- Book Reviews
  - Retrieval
- Needs
  - Registration
  - Evaluation
  - Registration

Book Catalogs
- Delete
- Update
- Search
- Mark up
- Registration

XML DB Handler
- KB (book data)
- IRUN (need data)

Book catalogs
- Update time
- Current state
- Evaluated value
- User’s interest and needs
A). Direct Storage of XML

XML data can be directly stored in XML DB.

Amazon Web Service

<Book>
  <Catalog/>
</Book>

RDB

XML DB
B). Semi-structured Data Handling

The structure of book data is different from book to book.

<Book>
<Catalog/>
<Comment/>
</Book>
C). Frequent Structural Change

Relational DB:

XML DB:
Content - 1. Web Document

- <BookShelf>
  - <Book>
    + <Image>
      <ASIN>0201702525</ASIN>
    + <Title>
      <AverageRating>4.5</AverageRating>
    + <Author>
    + <DetailPageURL>
    <Price>$44.99</Price>
    + <Publisher>
    </Publisher>
    <PublicationDate>2001-11-24</PublicationDate>
    <NumberOfPages>504</NumberOfPages>
  - <Update>
    <Latest>20071224185048</Latest>
    <Added>20071214182858</Added>
    <Commented>20071224184913</Commented>
    <Recommended>20071224185048</Recommended>
    <Search>20071214183018</Search>
  </Update>
  <SalesRank>459835</SalesRank>
  <State>Recommended</State>
  - <Extended>
    - <Comment>
      <![CDATA[
        Good book for software developer
      ]]>}
    </Comment>
  </Extended>
</Book>
</BookShelf>

**Update information:**
- Added time
- Commented time
- Recommended time
- Searched time

**Current state** of a book

**Comment** added by a user.
Semi-structured Data

• Web ←→ Data

• Information interchage, exchange → document structure

• Semi-structured Data
• `{ name: ”Alan”, tel: 2157786, email: ”abc@wwexch.net” }`
Web Data - Labels

• Duplicate labels

  { name: "Alan", tel: 2157786, tel: 3782535 }

• Many labels or missing labels

  { person:
    {name: "Alan", tel:2157786, email: bc@wwexch.net"},
    person:
    {name: {first: "Sara", last:"Green"},
      tel: 2136877, email: "sara@the.xyz.edu"},
    person:
    {name: "Fred", tel: 4257783, Height: 183 }
  }
A relation and its XML form

Fruits-table = fruit-name, string(6), color, string(5)
[ Apple, Green]
[ Apple, Red ]
• <?XML VERSION ="1.0" STANDALONE = "YES"?>
  <Fruits-table>
    <FRUITS>
      <FRUIT> <NAME> Apple <\NAME>
               <COLOR> Green <\COLOR>
      </FRUIT>
      <FRUIT> <NAME> Apple <\NAME>
               <COLOR> Red <\COLOR>
      </FRUIT>
    </FRUITS>
  </Fruits-table>
SQL Extensions (SQL 2003)

- `xmlelement → creates XML elements`

- `xmlattributes → creates attributes`

```sql
select xmlelement ( name "account,
    xmlattributes (account_number as account_number),
    xmlelement ( name "branch_name",branch_name),
    xmlelement ( name "balance",balance))
from account
```
• SQL 2003 → nested XML output
• Each tuple → XML element

```xml
<bank>
  <account>
    <row>
      <account-number> A-101 </account-number>
      <branch-name> Downtown </branch-name>
      <balance> 500 </balance>
    </row>
    <row>
      more data .....
    </row>
  </account>
</bank>
```
Data in XML - SQL 2003

- Ability to specify new tags + create nested tag structures → XML is a way to exchange **data** (db) + documents.
  - XML → extensive use in data exchange applications

- 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>
```
Data in XML (new std. SQL2003)

- `<university-3>`
- `<department dept name="Comp. Sci.">`
  - `<building> Taylor </building>`
  - `<budget> 100000 </budget>`
- `</department>`
- `<department dept name="Biology">`
  - `<building> Watson </building>`
  - `<budget> 90000 </budget>`
- `</department>`
- `<course course id="CS-101" dept name="Comp. Sci."` instructors="10101 83821">
  - `<title> Intro. to Computer Science </title>`
  - `<credits> 4 </credits>`
- `</course>`
- `....`
- `<instructor IID="10101" dept name="Comp. Sci.">`
  - `<name> Srinivasan </name>`
  - `<salary> 65000 </salary>`
- `</instructor>`
- `....`
- `</university-3>`
## 1. Contents - web documents

<table>
<thead>
<tr>
<th>Web document</th>
<th>Semi-structured data</th>
<th>Web query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Web documents</td>
<td>Semi-structured data</td>
<td>Web mining</td>
</tr>
<tr>
<td>Web structure and links</td>
<td>Structured data</td>
<td>Web mining</td>
</tr>
<tr>
<td>Web Usage logs and tables</td>
<td>Structured data</td>
<td>Web mining</td>
</tr>
</tbody>
</table>
Summary - 1

1. Content model → usage, interface, query ← Users

2. Representation
   ← 1. storage level
   → 2. content level

3. XML → widely researched and supported → authoring, editing, parsing, ....
Summary -2

1. XML query tools
   → xpath; xquery; xslt (all use xpath)
   → tree / arbitrary graph

2. SQL → can query GIS data and relational data
   (XML converted to relational form)

3. Query Interfaces → Type A and Type B

4. EHRs → AQL (uses SQL structure + XML addresses) ; XML templates
Summary - 3

1. SQL for map data

2. a) XML, b) XML query languages, c) Berkeley DB XML (free download)

3. Web Services and Web Resources

4. Recent increase in research activity → “New Query Language Interfaces”

5. High-level user interfaces
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
XML Examples

- Internet - RSS, ATOM
  - XHTML, Web Service Formats: SOAP, WSDL

File Format: Microsoft Office, Open Office, Apple’s iWork

Industrial - Insurance (ACORD),
  - Clinical Trials (cdisc)
  - Financial (FIX, FpML)

- Many Applications use XML - Storage or Data exchange
XML Document

```xml
<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>
```
Research Issues

• 1. Data \( \rightarrow \) Chemistry structures, EHRs
• \( \rightarrow \) Structural information is captured in tree model or graph model for querying

2. Graph is more flexible
3. Tree model is simple \( \rightarrow \) Single root, no cycle, unique path from root to a leaf.
   Graph \( \rightarrow \) pointer to ancestor and descendents
4. Semi-structured Data \( \rightarrow \) schema sharing
Old Data Models - List Processing

• 1. Hierarchical Model - Tree (rooted, acyclic, unique path from root to leaf)

• 2. Network Model - Linked list

1) and 2) → influenced by list structure
Content: Table - Set/bag (represent as?)

company section employee

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>s1</td>
<td>e1</td>
</tr>
<tr>
<td>c1</td>
<td>s1</td>
<td>e2</td>
</tr>
<tr>
<td>c1</td>
<td>s2</td>
<td>e3</td>
</tr>
</tbody>
</table>

- <company id="c1">
  - <section id="s1">
    - <employee id="e1"/>
    - <employee id="e2"/>
  - </section>
  - <section id="s2">
    - <employee id="e3"/>
  - </section>
- </company>
Data in Tree (list)

<employeeList>
  <employee id="e1">
    <company id="c1"/>
    <section id="s1"/>
  </employee>
  <employee id="e2">
    <company id="c1"/>
    <section id="s1"/>
  </employee>
  <employee id="e3">
    <company id="c1"/>
    <section id="s2"/>
  </employee>
</employeeList>
Relational Model (set)- EF Codd 1971 (IBM)

• A. Two levels-
  1. User → Sets and set operations

---------------------------------------

2. Storage → list;
  • -User [need elements]; (no navigation)
    -Storage [store over list; provide thru index or list search]
  • User [need set operations]
    ← do them on your own
Table form $\rightarrow$ set (product set)

<table>
<thead>
<tr>
<th>company</th>
<th>section</th>
<th>employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>s1</td>
<td>e1</td>
</tr>
<tr>
<td>c1</td>
<td>s1</td>
<td>e2</td>
</tr>
<tr>
<td>c1</td>
<td>s2</td>
<td>e3</td>
</tr>
</tbody>
</table>

- Table form of data $\rightarrow$ set, or bag
- Operations $\rightarrow$ set operations
  $\rightarrow$ Query language
Comparison of methods

- Old Models - Hierarchical Model
  → variation over list structure
  → Started from Bottom: Query on list

Network Model → variation over list

Relational Model: Top Down Approach →
Set + Set operations: Two layers
→ No navigation as in old models
→ influence over query operations
XML - Most Recent Innovations

• Can be a Tree with UNIX directory style paths

• Can maintain redundant IDs to know the linked information
Application Design and Development

- Application Programs and User Interfaces
- Tree Structured Data
  (a) Clint-side - XHTML, CSS, JavaScript
  (b) Storage side - RDBMS, XML, JSON, YAML, DOT, ...
- Web Fundamentals - Client + web server + Database Server
- Servlets and JSP, PHP
- Application Architectures
- Rapid Application Development
- Application Performance
- Application Security
- Encryption and Its Applications
Data Interchange

• Program 1 $\rightarrow$ CSV (comma Separated values)
• Program 2 $\leftarrow$ CSV values

• Program data dump
• Stacks
• ORACLE Database Dump
• Arrays
• Abstract Data Types

• PROGRAMMING $\rightarrow$ to upload and process
• $\rightarrow$ knowledge of syntax and semantics

• NEW TRENDS (data sharing among multiple applications)
  $\rightarrow$ YAML, Jason, XML, Candle
Information Interchange

- Information System 1 → Amazon Java books
- Information Systems 2 ← Amazon Books

1. Objects - Books, Rooms with id and (x,y) coordinates, Students, Courses, ....
2. Documents - web documents, financial statements of companies, ...
3. Graphs and structures - Proteins, Maps, ...
4. Information → Sets (RDB), relation! DB!
5. Tree-structured Data (XML), ...

→ Syntax and Semantics

Tree Structured Data → XML, JSON, Candle Markup
XML - STYLE MARKUP LANGUAGES

**Data Mark-up**: Configuration files, Internet Messages, Sharing Data and Objects between programming Languages

**Document Mark-up**: Web Documents, Database contents

**Purpose**: Exchange of data or exchange of documents, Storage

**YAML** → cross language, Unicode based, **data** serialization language (Data Mark-up)

**Candle Mark-up** → (Document mark-up for static data)

The syntax is based on XML, but have many differences
YAML

**Designed** → common data types of different programming languages.

**Superset** → JSON (YAML Version 1.2)

**Goals:**
1. easily readable by humans.
2. portable between programming languages.
3. matches the [native data structures](#) of most programming languages.
4. has a consistent model to support generic tools.
5. supports one-pass processing.
6. expressive and extensible.
7. is easy to implement and use.
YAML integrates and builds upon concepts
    (many tools + Software)
described by C,
    Java,
    Perl, Python, Ruby,
RFC0822 (MAIL),
RFC1866 (HTML),
RFC2045 (MIME),
RFC2396 (URI),
XML, SAX, SOAP, and
JSON.

Reference:  http://www.yaml.org/spec/1.2/spec.html  ( many more)
Candle Markup → Document Markup
→ Can do Data Markup easily
→ is an ideal format for general-purpose data serialization.
→ It works well for both structured object data and mixed text content.
→ It has a terse and readable syntax, as well as,
→ a clean and strongly-typed data model,
→ It is better than many existing textual serialization formats: XML, JSON, YAML.

→ Candle Markup is a subset of the Candle language
→ used as a document format for static data.
→ The syntax of Candle Markup is designed based on XML
Example ( XML )

```xml
<menu id="file" value="File">
    <popup>
        <menuitem value="New" onclick="CreateNewDoc()" />
        <menuitem value="Open" onclick="OpenDoc()" />
        <menuitem value="Close" onclick="CloseDoc()" />
    </popup>
</menu>
```

Example ( JSON )

```json
{"menu": {
    "id": "file", "value": "File",
    "popup": {
        "menuitem": [
            {"value": "New", "onclick": "CreateNewDoc()"},
            {"value": "Open", "onclick": "OpenDoc()"},
            {"value": "Close", "onclick": "CloseDoc()"}
        ]
    }
}}
```
Candle Object Notation (comparison with JSON):

→ objects have explicit name (instead of encoding it as key string);
→ attribute name does not need to be double quoted;
→ There's no need of delimiter, like comma, between the attributes.
DOT (graph description language)

- example script that describes the bonding structure of an ethane molecule. This is an undirected graph and contains node attributes.

```
graph ethane {
  C_0 -- H_0 [type=s];
  C_0 -- H_1 [type=s];
  C_0 -- H_2 [type=s];
  C_0 -- C_1 [type=s];
  C_1 -- H_3 [type=s];
  C_1 -- H_4 [type=s];
  C_1 -- H_5 [type=s];
}
```

- Many interfaces for graphic visualization and query
Conclusions

• 1. Information Interchange $\rightarrow$ is common

• 2. ADTs $\rightarrow$ objects with schema details
   $\rightarrow$ Languages (XML, JSON, ....)

• 3. Storage $\rightarrow$ Transform
   $\rightarrow$ Query