Logistics

- Lab 1 – data preparation
  - Instructions posted
  - In-class Lab Friday (XML, JSON, HTML)
  - Homework Due Friday 11:59pm

- Piazza Signup

Review

- Data Types and Sources
  - Tabular
  - Semi-structured
  - Text, video, images
  - Graph data
- Data Models
- Data Preparation

Data Model and Schema

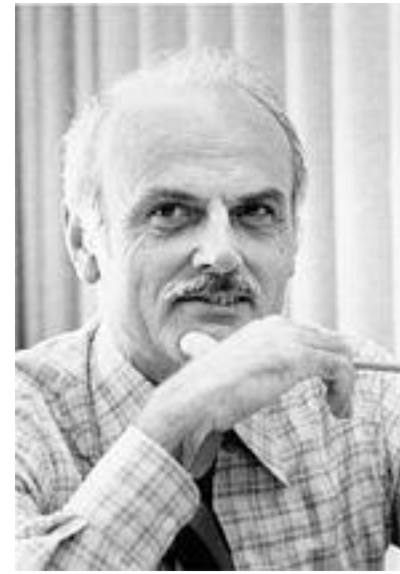
A ***data model*** is a collection of concepts for describing data.

A ***schema*** is a description of a particular collection of data, using a given data model.

Some common data models

- The relational model of data is the most widely used for record keeping.
  - Main concept: relation, basically a table with rows and columns.
  - Every relation has a schema, which describes the columns
- Semi-structured models in increasing use (e.g. XML)
  - Main concept: self-describing (tagged) document, basically a textual hierarchy (tree) of labeled values
  - Document Type Definition (DTD) or Schema possible, but not required
- Free text (and hypertext) widely used as well
  - Data represented for human consumption
    - Visual aspects and linguistic subtlety more important than clearly structured data
- Others: RDF Triple, Graph, Streaming, Probabilistic Data, Key-Value, Array/Matrix, Column Stores

The Relational Model*



E. F., “Ted” Codd
Turing Award 1981

- A Data Model based on Set/Bag Theory
  – Support Relational Algebra
- The Relational Model is Ubiquitous:
  - MySQL, PostgreSQL, Oracle, DB2, SQLServer, …
  - Foundational work done at
    - IBM - System R
    - UC Berkeley - Ingres
- Object-oriented concepts have been merged in
    - Early work: POSTGRES research project at Berkeley
    - Informix, IBM DB2, Oracle 8i

**Codd, E. F. (1970). "A relational model of data for large shared data banks". Communications of the ACM 13 (6): 37*

Relational Database: Definitions

- *Relational database:* a set of *relations*
- *Relation:* made up of 2 parts:
  - *Schema* : specifies name of relation, plus name and type of each column

    Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real)
  - *Instance* : the actual data at a given time
    - #rows = *cardinality*
    - #fields = *degree / arity*

Ex: Instance of Students Relation

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith @math	19	3.8

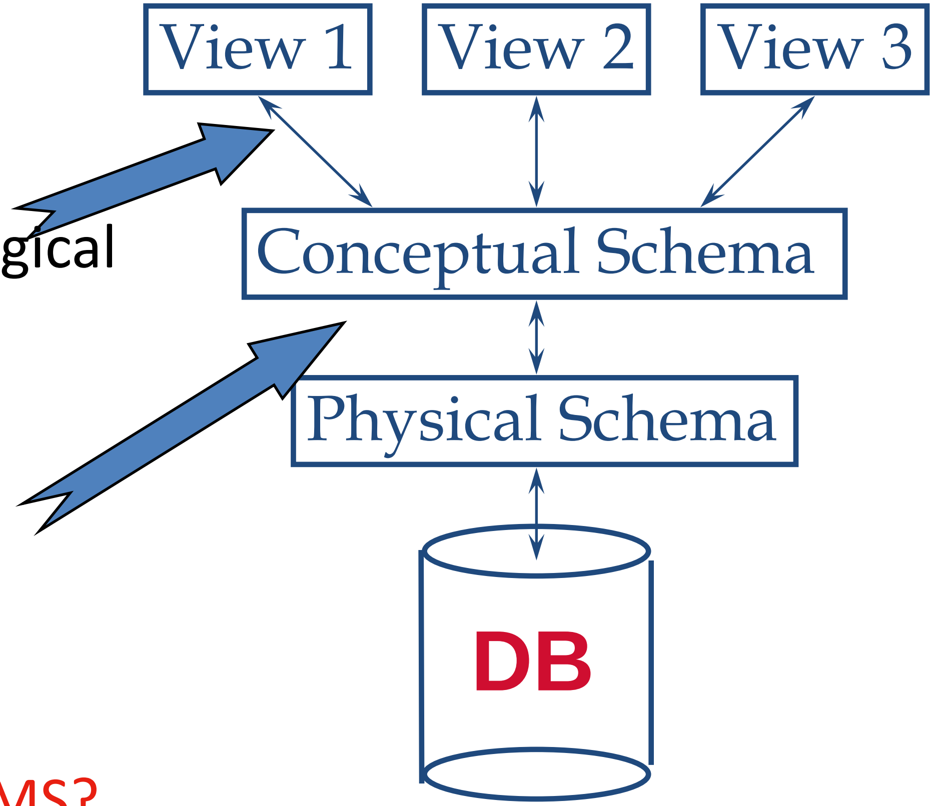
- Cardinality = 3, arity = 5 , all rows distinct

- The relation is true for these tuples and false for others (a.k.a, the closed world assumption)

Levels of Abstraction

- Applications insulated from how data is structured and stored.
- Logical data independence: Protection from changes in logical structure of data.
- Physical data independence: Protection from changes in physical structure of data.
- Q: Why are these particularly important for DBMS?

  Because rate of change of DB applications is incredibly slow.



Other Table-like Data Models: Pandas/Python

- **Series**: a named, ordered array/dictionary
  - Values can be any Numpy data type object
  - The keys of the dictionary are the **indexes**
  - Built on NumPy's **ndarray**

- **DataFrame**: a table with named columns
  - Represented as a map Dict (col_name -> series)
  - Each Series object represents a column

- SQL vs. Pandas / SQL + Pandas

Cases where Tables break as a data model? E.g., semi-/un-structured data

- Too limited in structure?
- Too rigid?
- Too costly up front?
- Examples… ??

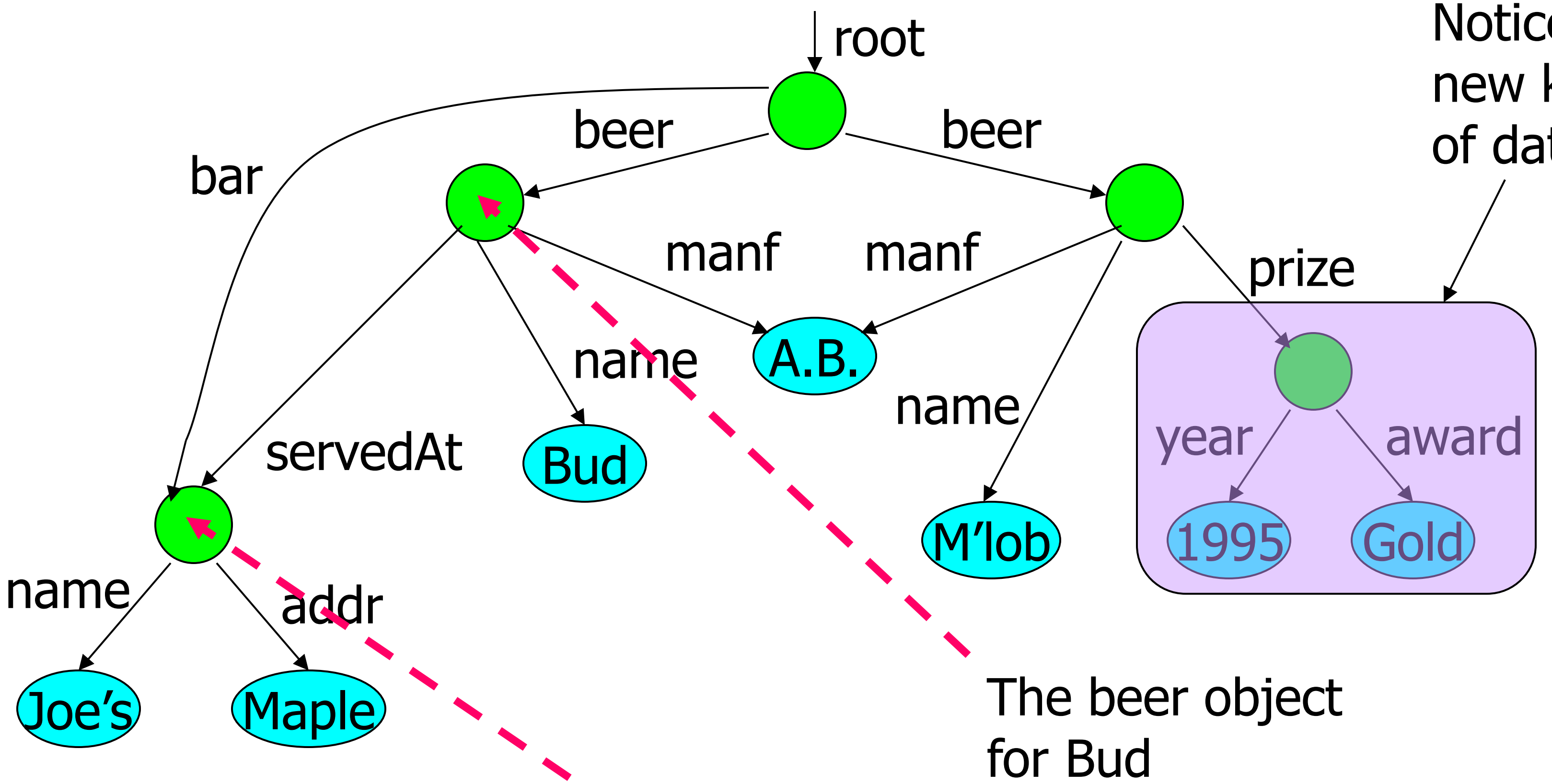
Semistructured Data

- Another data model, based on trees.
- Motivation: flexible representation of data.
  - Often, data comes from multiple sources with differences in notation, meaning, etc.
- Motivation: sharing of *documents* among systems and databases.

Hierarchical Semistructured Data

- Nodes = objects.
- Labels on arcs (attributes, relationships).
- Atomic values at leaf nodes (nodes with no arcs out).
- Flexibility: no restriction on:
  - Labels out of a node.
  - Number of successors with a given label.

Example: Data Graph



Notice a new kind of data.

The beer object for Bud

The bar object for Joe's Bar

Well-Formed and Valid XML

- *Well-Formed XML* allows you to invent your own tags.
  - Similar to labels in semistructured data.

- *Valid XML* involves a DTD (*Document Type Definition*), a grammar for tags.

Well-Formed XML

- Start the document with a *declaration*, surrounded by <?xml … ?> .
- Normal declaration is:

```
<?xml version = "1.0" standalone
  = "yes" ?>
```

- Balance of document is a *root tag* surrounding nested tags.

Tags

- Tags, as in HTML, are normally matched pairs, as <FOO> … </FOO> .
- Tags may be nested arbitrarily.
- XML tags are case sensitive.

Example: Well-Formed XML

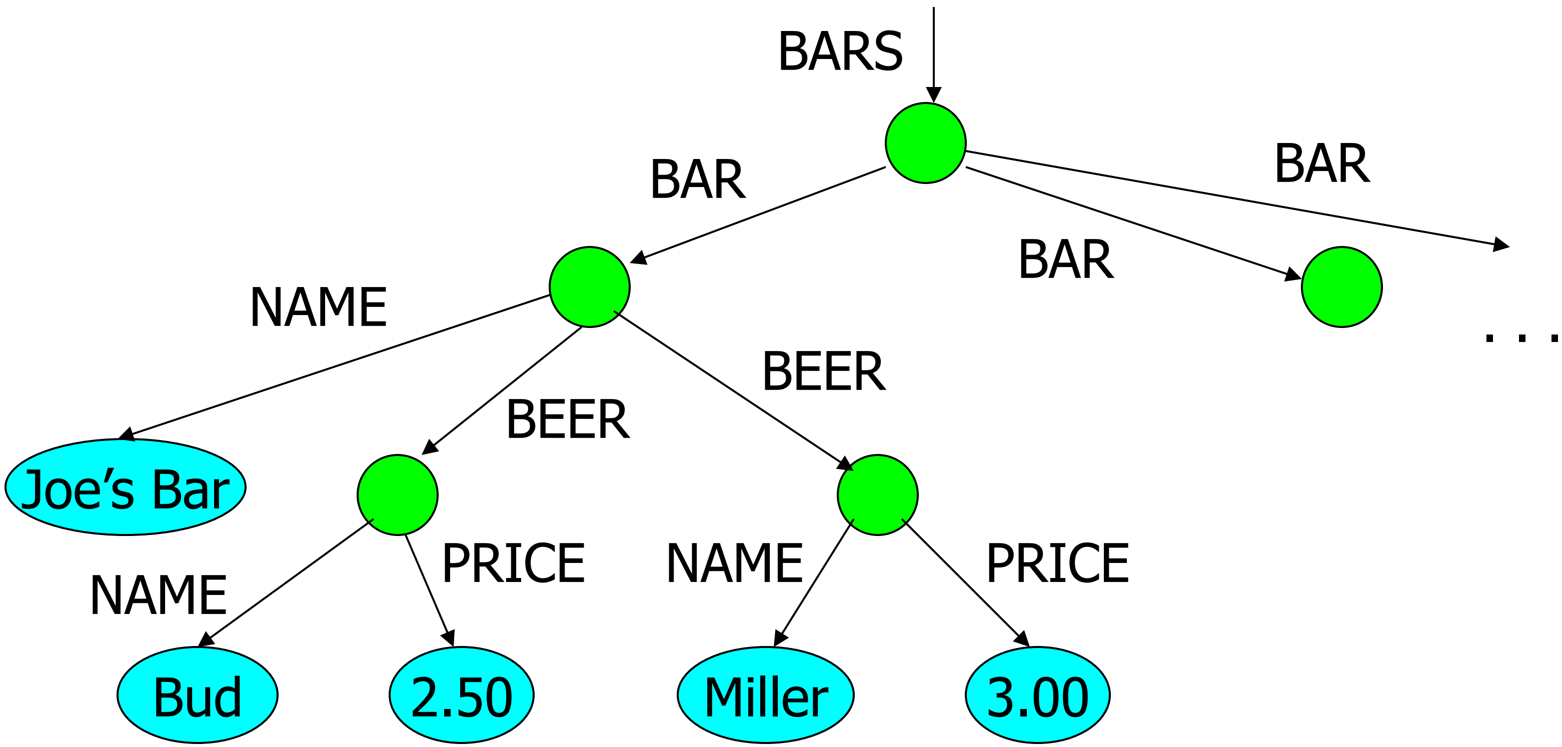
```
<?xml version = “1.0” standalone = “yes” ?>
<BARS>
    <BAR><NAME>Joe’s Bar</NAME>
     <BEER><NAME>Bud</NAME>
          <PRICE>2.50</PRICE></BEER>
     <BEER><NAME>Miller</NAME>
          <PRICE>3.00</PRICE></BEER>
    </BAR>
    <BAR> …
</BARS>
```

A NAME subobject

A BEER subobject

Example

- Well-Formed XML with nested tags is exactly the same idea as trees of semistructured data.
- The <BARS> XML document is:



XML Schema: DTD Structure

```
<!DOCTYPE <root tag> [
   <!ELEMENT <name> (<components>) >
   . . . more elements . . .
]>
```

XML Schema in XSD

```
<location>
  <latitude>37.78333</latitude>
  <longitude>122.4167</longitude>
</location>
```

An XML schema for this element:

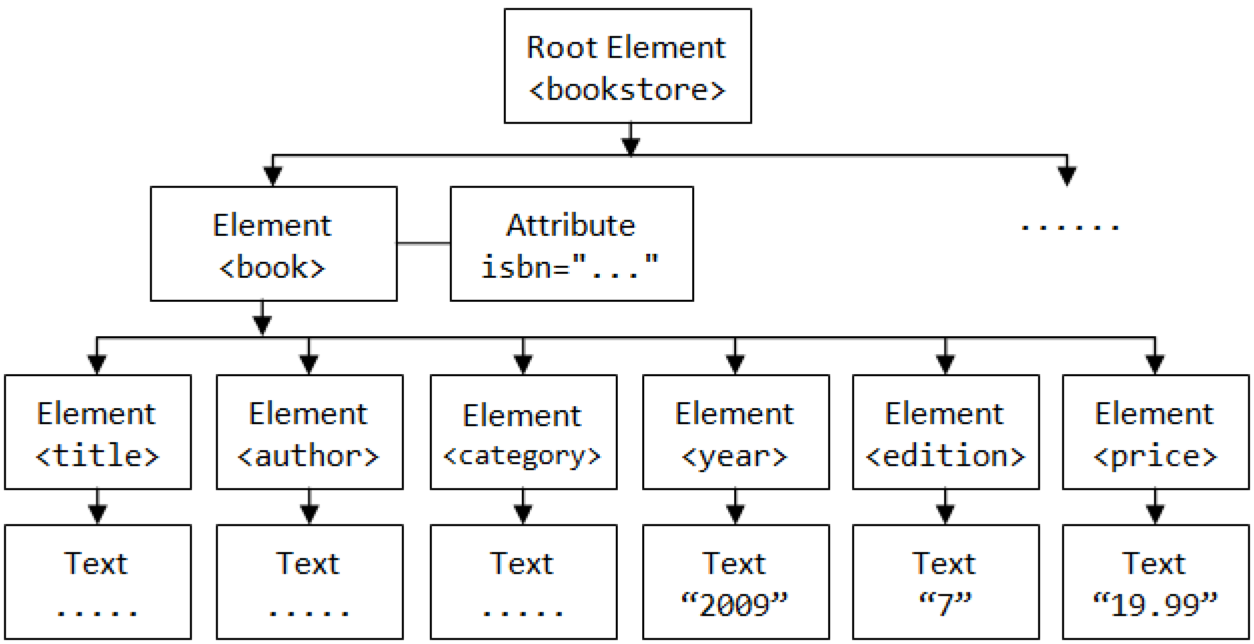
…

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified">
<xsd:complexType name="location">
  <xsd:sequence>
    <xsd:element name="latitude" type="xsd:decimal"/>
    <xsd:element name="longitude" type="xsd:decimal"/>
  </xsd:sequence>
</xsd:complexType name=“location">
```

XML and DOM

XML is a text format that encodes DOM (Document-Object Models) which is a data structure e.g. for Web pages.

The DOM is tree-structured:



JSON format --> tree structure

```
{ "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
   "age": 25,
   "height_cm": 167.6,
   "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3100"
    }
}
```

Prerequisites for “Schemaless” DBs

- Need **external** and **internal** representations for all data types that will be used.
- **Internal:** a dynamically-typed, object-oriented language (like Java)
- **External:** an extensible data description language: JSON or XML

- **For Performance:** Fast SerDe (Serialization and DeSerialization) so internal data structures can be efficiently pushed or extracted from disk or network.

Prerequisites for “Schemaless” DBs

- JSON includes named fields in a tree structure. Primitive types (e.g. string, number, boolean,…) are implicit.

- We can read JSON data (or XML) and automatically create internal representations for complex data.

- Using the field names and object structure, we can query these objects once loaded.

XML vs. JSON

XML:

- Separation between schema and data.
- Data can be represented and stored without schema (as strings).
- More verbose (but not true after compression or in DB).
- Standard Query/Transformation languages XSLT and Xquery.

JSON:

- Types inferred inline. Schema rarely used but can be.
- Data without schema use type inference (string, int, float,…).
- More succinct in ASCII form.
- Transformation/ingestion rely on code (Java or Javascript).

NoSQL Storage Systems

	Data Model
Cassandra	Columnfamily
CouchDB	Document
HBase	Columnfamily
MongoDB	Document
Neo4J	Graph
Redis	Collection
Riak	Document
Scalaris	Key/value
Tokyo Cabinet	Key/value
Voldemort	Key/value

CouchDB Data Model (JSON)

- “With CouchDB, no schema is enforced, so new document types with new meaning can be safely added alongside the old.”
- A CouchDB document is an object that consists of named fields. Field values may be:
  - strings, numbers, dates,
  - ordered lists, associative maps

"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."

RDF Basics

- RDF is based on the idea of identifying resources using **Web identifiers** and describing resources in terms of simple **properties** and property **values**.

- To identify resources, RDF uses **Uniform Resource Identifiers (URIs**) and **URI references (URIrefs).**

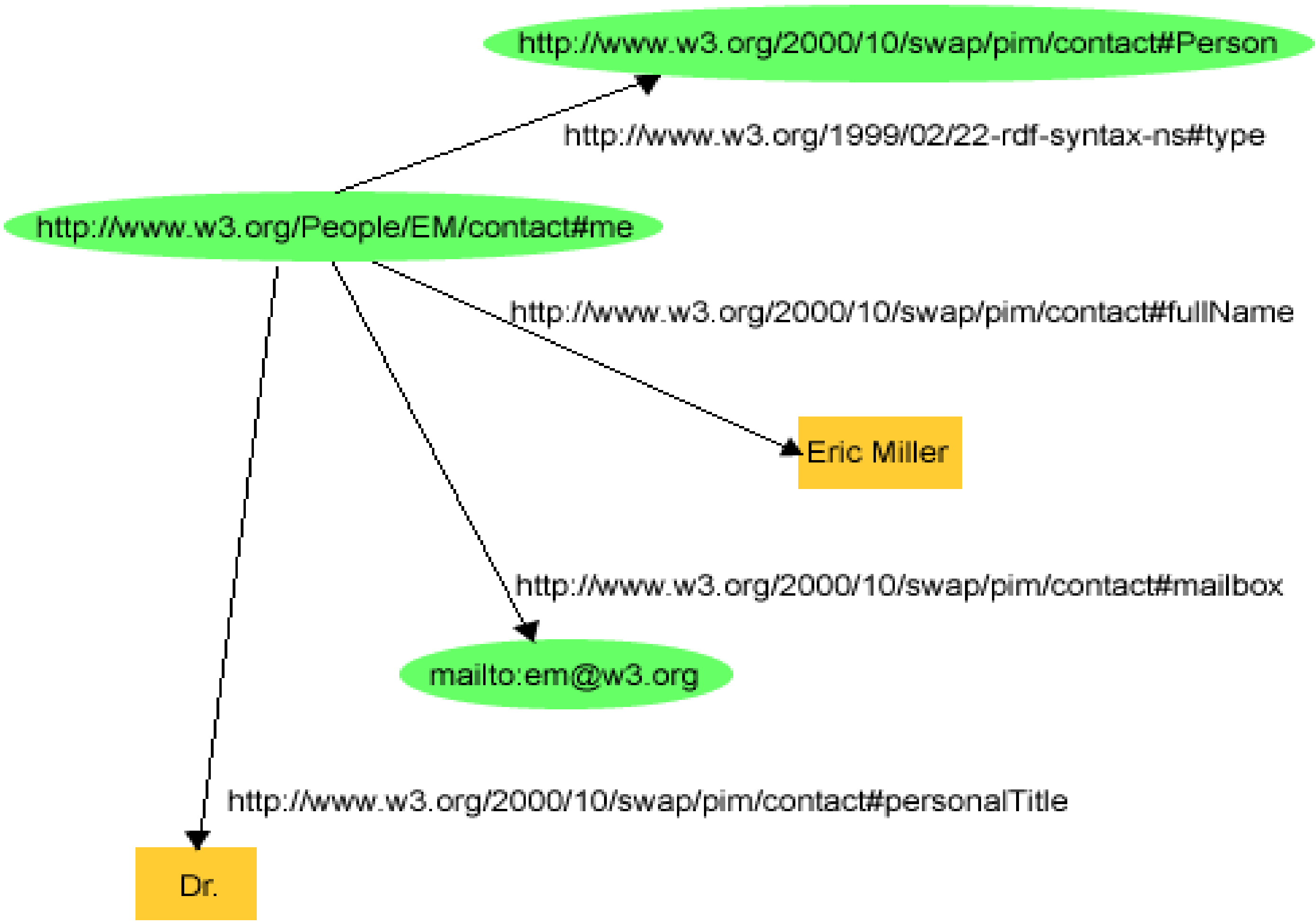
- **Definition:** A **resource** is anything that is identifiable by a URIref.

Example

- Consider the following information:

  "there is a Person identified by `http://www.w3.org/People/EM/contact#me`, whose name is Eric Miller, whose email address is em@w3.org, and whose title is Dr."

RDF Graph Example (cont'd)



Basics (cont'd)

- Forget the long URIs for the moment!

- RDF is based on the idea that **the resources being described have properties which have values**, and that resources can be described by making **statements**, similar to the ones above, that specify those properties and values.

- **Terminology:**
  - The part that identifies the thing the statement is about is called the **subject**.
  - The part that identifies the property or characteristic of the subject that the statement specifies is called the **predicate**.
  - The part that identifies the value of that property is called the **object**.

Example

`http://www.example.org/index.html` has a creator whose value is “John Smith”

- The **subject** is the URL `http://www.example.org/index.html`
- The **predicate** is the word "creator"
- The **object** is the phrase “John Smith”

RDF Triples → RDF Graphs (SPARQL)

- RDF statements can be written down using **triple notation.** In this notation, a statement is written as follows:

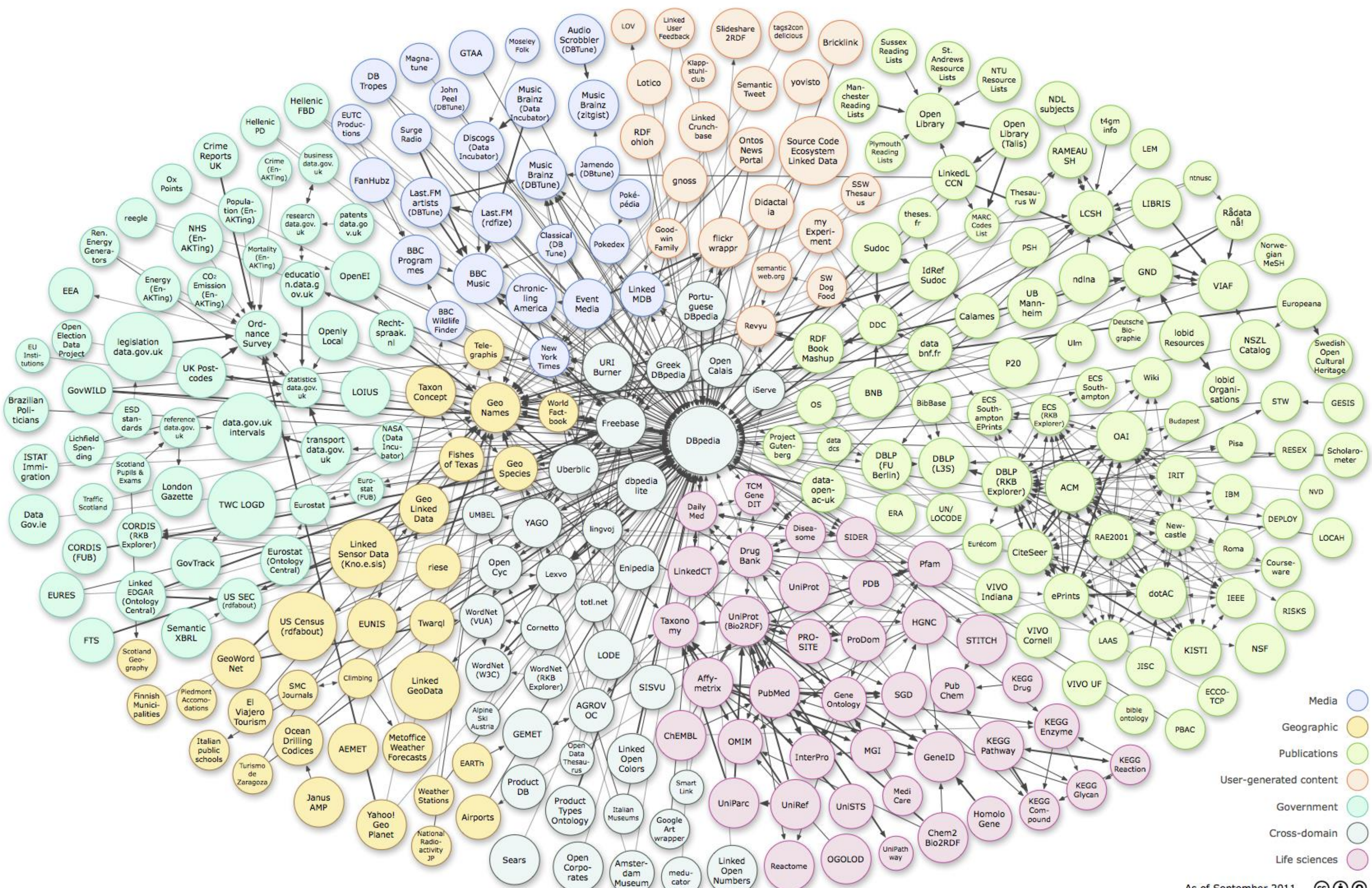
  subject predicate object .

- **Example:**

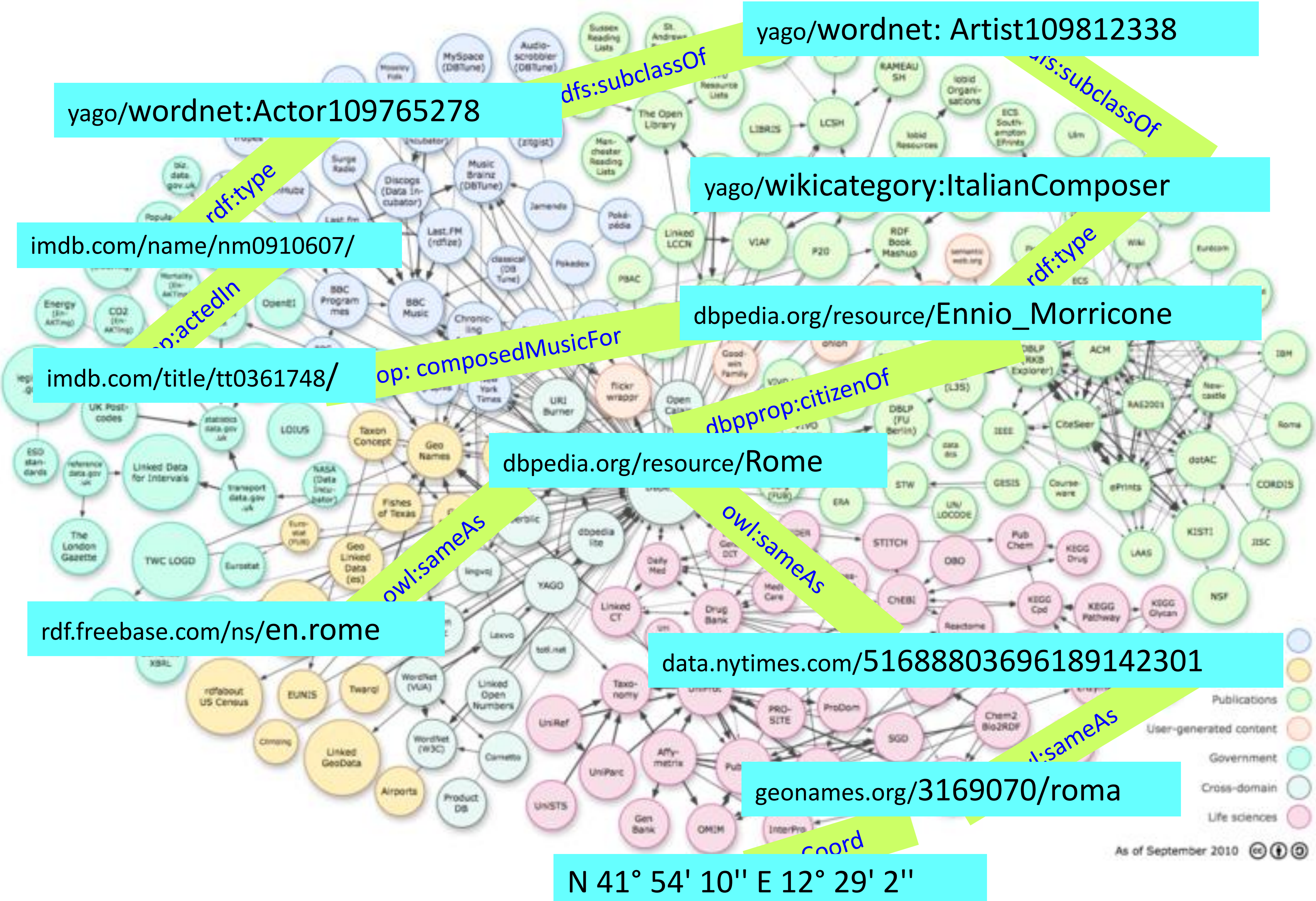
```
<http://www.example.org/index.html>
   <http://purl.org/dc/elements/1.1/creator>
   <http://www.example.org/staffid/85740> .

<http://www.example.org/index.html>
   <http://www.example.org/terms/creation-date> "August 16, 1999" .

<http://www.example.org/index.html>
   <http://purl.org/dc/elements/1.1/language> "en" .
```

- **Note:** In this notation URIs are written out completely, in angle brackets.

RDF Triples on the Web



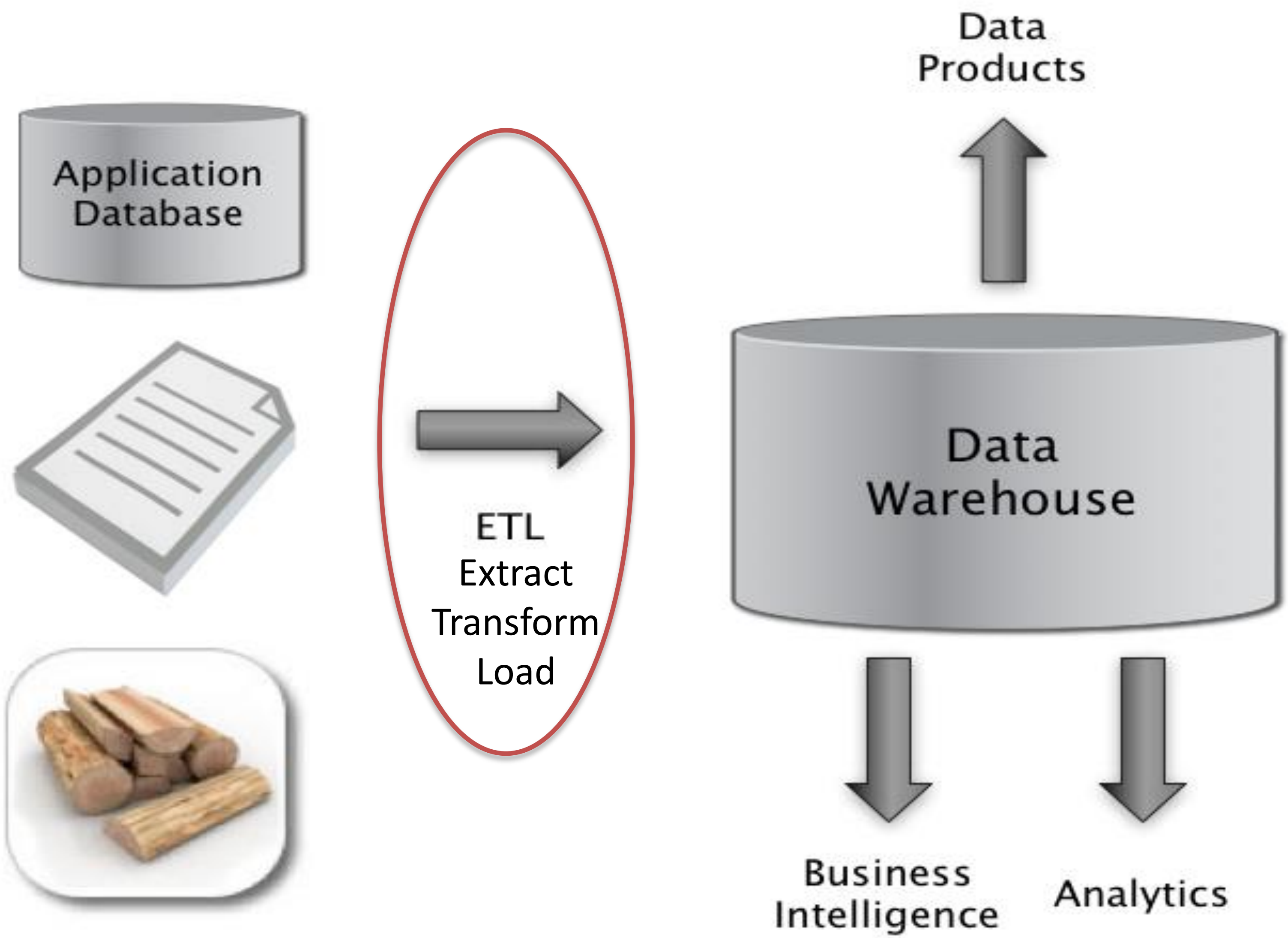
LOD: Linked RDF Triples on the Web



Outline

- Data Types and Sources
- Data Model
- Data Preparation
  - Big Picture

The Big Picture



What are your…

- Data Sources?
- ETL Process/Workflow and tools?
- Data Warehouse?
- Business Intelligence and Analytics?

The Businessperson

- Data Sources
  - Web pages
  - Excel
- ETL
  - Copy and paste
- Data Warehouse
  - Excel
- Business Intelligence and Analytics
  - Excel functions
  - Excel charts

The Programmer

- Data Sources
  - Web scraping, web services API
  - Excel spreadsheet exported as CSV
  - Database queries
- ETL
  - wget, curl, Beautiful Soup, lxml
- Data Warehouse
  - Flat files
- Business Intelligence and Analytics
  - Numpy, Matplotlib, R, Matlab

The Enterprise

- Data Sources
  - Application databases
  - Intranet files
  - Application server log files
- ETL
  - Informatica, IBM DataStage, Ab Initio, Talend
- Data Warehouse
  - Teradata, Oracle, IBM DB2, Microsoft SQL Server
- Business Intelligence and Analytics
  - Business Objects, Cognos, Microstrategy
  - SAS, SPSS, R

The Web Company

- Data Sources
  - Application databases
  - Logs from the services tier
  - Web crawl data
- ETL
  - Flume, Sqoop, Pig, Crunch, Oozie
- Data Warehouse
  - Hadoop/Hive, Spark/Shark
- Business Intelligence and Analytics
  - Custom dashboards: Argus, BirdBrain
  - R

Data Preparation

- ETL
  - We need to **extract** data from the **source(s)**
  - We need to **load** data into the **sink**
  - We need to **transform** data at the source, sink, or in a **staging area**

  - Sources: file, database, event log, web site, HDFS…
  - Sinks: Python, R, SQLite, RDBMS, NoSQL store, files, HDFS…

Data Preparation

- Workflow
  - The **pipeline** often consists of many steps
    - Using Unix command line pipes and filters
    - head -50 wc_day6_1.log | cut -d ' ' -f 7 | sort | uniq -c | tail -10
    - Simple data transformation using cut, sort, regex, sed, awk, etc.
  - Hands-on experience in Lab1
  - If the workflow is to be used more than once, it can be **scheduled**
    - Scheduling can be time-based or event-based
    - Use publish-subscribe to register interest (e.g. Twitter feeds)
  - Recording the execution of a workflow is known as capturing **lineage** or **provenance**

HTML Tools - Parsing

- "Beautiful Soup"
  http://www.crummy.com/software/BeautifulSoup/
  a Python API for handling real HTML. DOM or SAX interfaces.

- "TagSoup"
  http://ccil.org/~cowan/XML/tagsoup/
  provides a Sax interface, i.e. a streaming parse, to Java applications. Can transform to a format you want using XSLT.

- Taggle, part of the Arabica toolset
  http://www.jezuk.co.uk/cgi-bin/view/arabica/code
  is a version of TagSoup written in C++. You may want to use this if you have a lot of data.

Event-Driven Parsing: SAX

<?xml version="1.0" encoding="UTF-8"?> Document Header

<!-- bookstore.xml --> Comment

<bookstore> Start-element “bookstore”

<book ISBN="0123456001"> Start-element “book”

<title>Java For Dummies</title> Start-element “title”

End-element “title”

<author>Tan Ah Teck</author>

<category>Programming</category>

<year>2009</year>

<edition>7</edition>

<price>19.99</price>

</book> End-element “book”

Event-Driven Parsing: SAX

A SAX parser finds all the open-close-tag events in an XML documents, and does callbacks to user code.

- User code can respond to only a subset of events corresponding to the tags it is interested in.
- User code can correctly compute aggregates from the data rather than create a record for each tag.
- User code must implement a state machine to keep track of “where it is” in the DOM tree.
- User code can implement flexible error recover strategies for ill-formed XML.

- Python XML parser: xml.sax, lxml

What about JSON?

Most JSON parsers construct the “DOM” directly.

A few JSON (SAX-style) parsers:

- json, iJSON – Python
- Jackson – Java

Summary

- Data Types and Sources
  – Schema on Read vs. Schema on Write
- Data Models
  – Set, HashTable, Tree, Graph..
- Data Preparation

- Unix data preparation lab 1 this Friday