§2. Models

Qin Zhang
Models

1. Hierarchical (IMS): late 1960s and 1970s
2. Network (CODASYL): 1970s
4. Entity-Relationship: 1970s
5. Extended Relational: 1980s
7. Object-oriented: late 1980s and early 1990s
8. Object-relational: late 1980s and early 1990s
9. XML
10. ...

Read “What Goes Around Comes Around” by Michael Stonebraker and Joseph M. Hellerstein
Different Types of Data

- **Structured data**
  All data conforms to a schema. Ex: business data

- **Semistructured data**
  Some structure in the data but implicit and irregular
  Ex: resume, ads

- **Unstructured data**
  No structure in data. Ex: text, sound, video, images
Different Types of Data

- **Structured data**
  All data conforms to a schema. Ex: business data

  **Focus of traditional databases**

- **Semistructured data**
  Some structure in the data but implicit and irregular
  Ex: resume, ads

- **Unstructured data**
  No structure in data. Ex: text, sound, video, images
Model 1: IMS

- **Hierarchical data model**: record types organized in a tree

  - **Supplier** (sno, sname, scity, sstate)
  - **Part** (pno, pname, psize, pcolor)
  - **Part** (pno, pname, psize, pcolor, qty, price)
  - **Supplier** (sno, sname, scity, sstate, qty, price)

- **Issues**
  - Record types must be arranged in a tree
  - Information is repeated
  - Existence depends on parents
  - Work with DL/1, a record-at-a-time language
  - (is this an inherent problem?)
Model 2: CODASYL

- **Networked data model**: record types organized in a network

  - Supplier (sno, sname, scity, sstate)
  - Part (pno, pname, psize, pcolor)

- **Issues**
  - Very complex; Programs must navigate the hyperspace
  - Load and recover as one gigantic object (the whole network)
  - Work with a record-at-a-time language
Model 2: CODASYL

- **Networked data model**: record types organized in a network

```
Supplier (sno, sname, scity, sstate)
```

```
Part (pno, pname, psize, pcolor)
```

```
Supply(qty, price)
```

- **Issues**
  - Very complex; Programs must navigate the hyperspace
  - Load and recover as one gigantic object (the whole network)
  - Work with a record-at-a-time language

- Any advantages?
Model 2: CODASYL

- **Networked data model**: record types organized in a network

  - **Supplier** (sno, sname, scity, sstate)
  - **Part** (pno, pname, psize, pcolor)

  - Supplies
  - Supplied by

  **Supply** (qty, price)

  - Any advantages?
  - Efficiency!

- **Issues**
  - Very complex; Programs must navigate the hyperspace
  - Load and recover as one gigantic object (the whole network)
  - Work with a record-at-a-time language
Model 3: Relational

You have already seen

– No specification of what storage looks like
  (In my personal opinion: structure is encoded in the tables / implicit links)
– Use set-at-a-time language: algebra or calculus

Any disadvantages?
Great Debate

- **Pro relational**
  - CODASYL is too complex
  - Trees/networks not flexible to represent common cases
  - Record-at-a-time languages are too hard to optimize
Great Debate

- **Pro relational**
  - CODASYL is too complex
  - Trees/networks not flexible to represent common cases
  - Record-at-a-time languages are too hard to optimize

- **Against relational**
  - COBOL programmers cannot understand relational languages
  - Impossible to implement the relational model efficiently
  - CODASYL can represent tables
Great Debate

- **Pro relational**
  - CODASYL is too complex
  - Trees/networks not flexible to represent common cases
  - Record-at-a-time languages are too hard to optimize

- **Against relational**
  - COBOL programmers cannot understand relational languages
  - Impossible to implement the relational model efficiently
  - CODASYL can represent tables

- Ultimately settled by the market place
Next we will discuss (and compare them with *relational*):
1. E/R Model
2. XML
Other models and NoSQL

- Next we will discuss (and compare then with *relational*):
  1. E/R Model
  2. XML

- NoSQL models.
  
  Including *key-value stores, document stores, graph DB systems* and *column store*
Next we will discuss (and compare then with *relational*):
1. E/R Model
2. XML

NoSQL models.
Including *key-value stores, document stores, graph DB systems* and *column store*

Big Data Models
1. I/O Model (SQL)  
2. Data Stream Model (NoSQL)
3. MapReduce and ActiveDHT (NoSQL)
Why NoSQL?

My personal opinions:
challenage in Big Data:
data types, storage and
computational issues
E/R Model and XML
(see separate slides)
NoSQL by Prof. Jennifer Widom:
https://www.youtube.com/watch?v=3pS1MF9mYJE&list=PLB142989A709CF0B8

https://www.youtube.com/watch?v=e5VLYZ_0tjg&list=PLB142989A709CF0B8&index=2
(Not in class)

Column Oriented Database
https://www.youtube.com/watch?v=mRvkikVuojU
Key-Value Stores

- Extremely simple interface
  - Data model: (key, value) pairs
  - Operations: Insert(key, value), Fetch(key), Update(key), Delete(key)
  - Some allow Fetch on range of keys

Looks like a dictionary :)

- Example systems
  - Google BigTable, Amazon Dynamo, Cassandra, Voldemort, HBase,
Document Stores

- **Like Key-Value Stores except value is document**
  - Data model: (key, document) pairs
  - Document: JSON, XML, other semistructured formats
  - Operations: **Insert**(key, document), **Fetch**(key), **Update**(key), **Delete**(key)
  - Also Fetch based on document contents

- **Example systems**
  - CouchDB, MongoDB, SimpleDB, ...
Data organized as a graph
- Data model: nodes and edges
- Nodes may have properties (including ID)
- Edges may have labels or roles

Reminds you CODASYL?
A Quick Summarization of NoSQL

- Target on specific tasks, use specific algorithms instead of SQL queries

- Not necessary to give the exact answers; Good approximations are often enough.

- Relax various requirements such as “consistency”
Now let’s talk about **efficiency**
§2.1 Input-Output Model
(see separate slides)
§2.2 Data Stream Model
(see separate slides)
§2.3 MapReduce and ActiveDHT (see separate slides)