CSE 544
Principles of Database Management Systems

Alvin Cheung

Fall 2015

Lecture 3 – Data models
A Never-Ending Story

Announcements

- Homework 1 is out
  - Due in 2 weeks
- Project
  - Please form teams this week and send us an email by next Monday
  - Start to think about class projects
  - More info on website (suggested topics will be posted)

References

- M. Stonebraker and J. Hellerstein. What Goes Around Comes Around. In "Readings in Database Systems" (aka the Red Book). 4th ed.

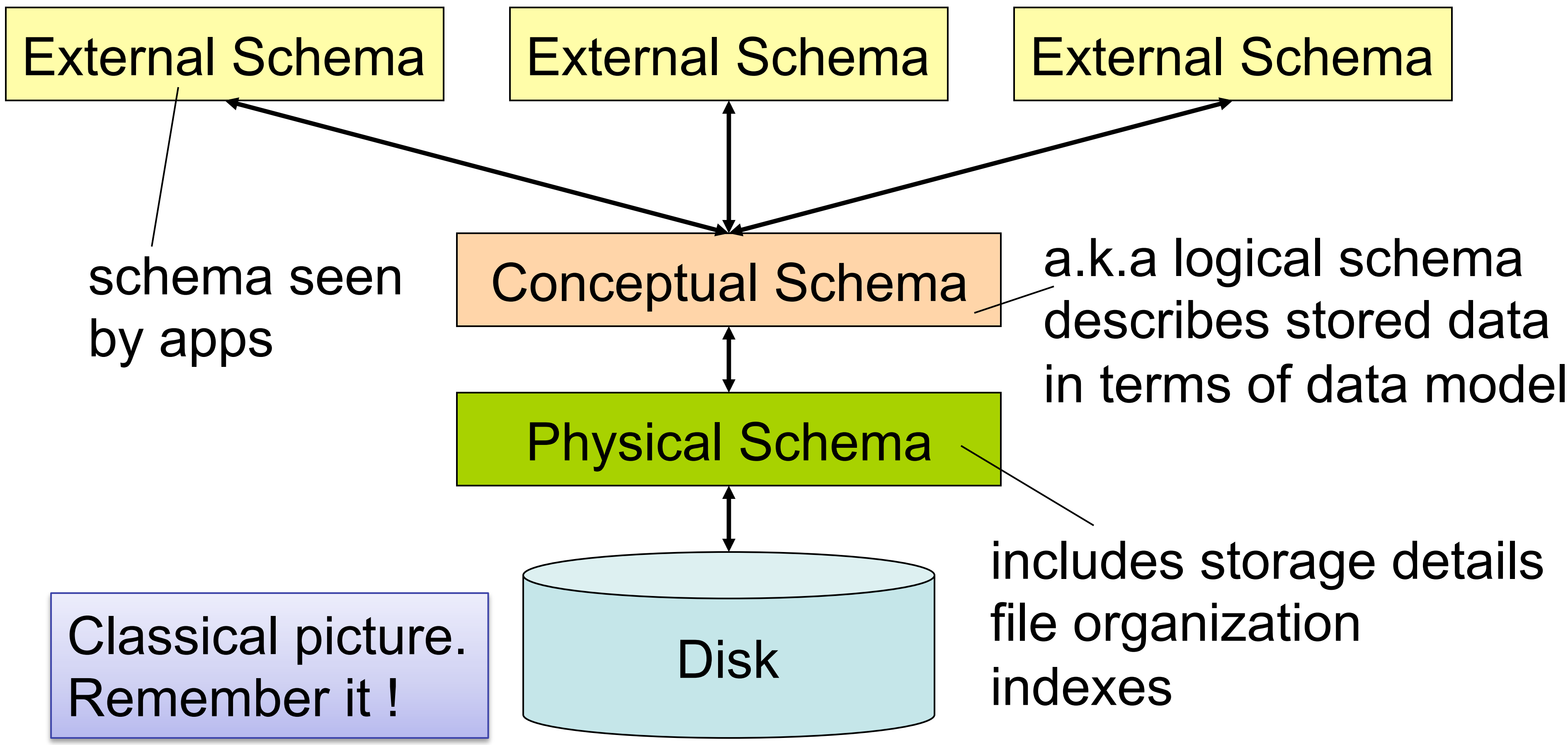
Data Model Motivation

- Applications need to model real-world data
  - Data typically includes entities and relationships between them
  - Example entities are students, courses, products, clients
  - Example relationships are course registrations, product purchases

- User somehow needs to define data to be stored in DBMS

- Data model enables a user to define the data using high-level constructs without worrying about many low-level details of how data will be stored on disk

Levels of Abstraction



Data Model Motivation (cont.)

DBMS needs to let user perform several tasks

- Define data that should live in the DBMS
  – Need a data model and a data definition language
- Ask questions about the data
- Insert, delete, and modify data
  – Need a data manipulation language

- Challenges
  – Want to minimize changes to applications when physical data layout changes (for performance) or logical structure of data changes (perhaps as database evolves over time)

Outline

- Different types of data
- Early data models
  - IMS
  - CODASYL
- Physical and logical independence in the relational model
- Data models that followed the relational model
- NoSQL data models

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

- Our focus: structured data & relational DBMSs

Outline

- Different types of data
- Early data models
  - IMS
  - CODASYL
- Physical and logical independence in the relational model
- Data models that followed the relational model
- NoSQL data models

Early Proposal 1: IMS

- What is it?

Early Proposal 1: IMS

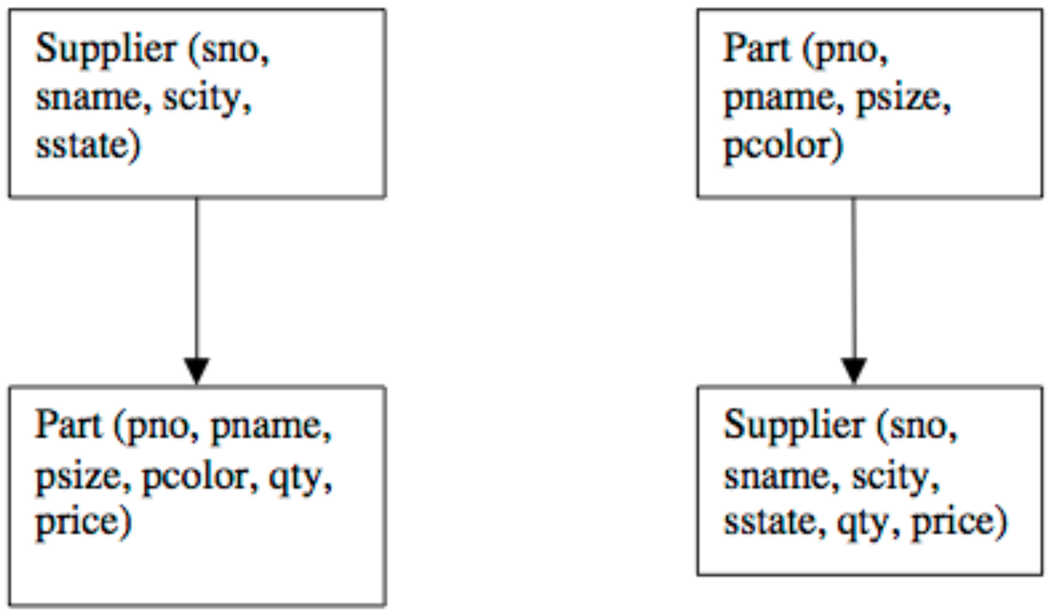
- **Hierarchical data model**

- **Record**
  - **Type**: collection of named fields with data types
  - **Instance**: must match type definition
  - Each instance must have a **key**
  - Record types must be arranged in a **tree**

- **IMS database** is collection of instances of record types organized in a tree

IMS Example

- Figure 2 from “What goes around comes around”



Data Manipulation Language: DL/1

- How does a programmer retrieve data in IMS?

Data Manipulation Language: DL/1

- Each record has a hierarchical sequence key (HSK)
  - Records are totally ordered: depth-first and left-to-right

- HSK defines semantics of commands:
  - get_next
  - get_next_within_parent

- **DL/1 is a record-at-a-time language**
  - Programmer constructs an algorithm for solving the query
  - Programmer must worry about query optimization

Data storage

- How is the data physically stored in IMS?

Data storage

- Root records
  - Stored sequentially (sorted on key)
  - Indexed in a B-tree using the key of the record
  - Hashed using the key of the record

- Dependent records
  - Physically sequential
  - Various forms of pointers

- Selected organizations restrict DL/1 commands
  - No updates allowed due to sequential organization
  - No “get-next” for hashed organization

Data Independence

- What is it?

Data Independence

- **Physical data independence**: Applications are insulated from changes in **physical storage details**

- **Logical data independence**: Applications are insulated from changes to **logical structure of the data**

- **Why are these properties important?**
  - Reduce program maintenance as
  - Logical database design changes over time
  - Physical database design tuned for performance

IMS Limitations

- **Tree-structured data model**
  - **Redundant** data, existence **depends on parent, artificial structure**
- **Record-at-a-time** user interface
  - User must specify **algorithm** to access data
- **Very limited physical independence**
  - Phys. organization limits possible operations
  - Application programs break if organization changes
- Provides **some logical independence**
  - DL/1 program runs on logical database
  - Difficult to achieve good logical data independence with a tree model

Early Proposal 2: CODASYL

- What is it?

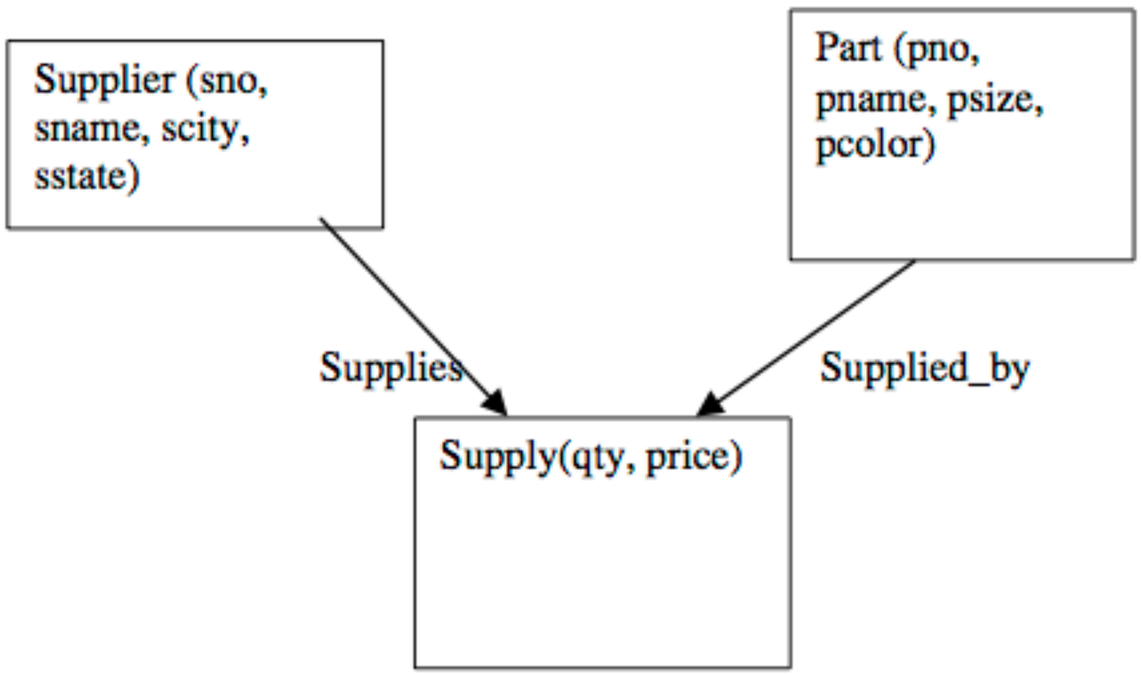
Early Proposal 2: CODASYL

- **Networked data model**

- Primitives are also **record types** with **keys**
- Record types are organized into **network**
  - A record can have multiple parents
  - Arcs between records are named
  - At least one entry point to the network
- Network model is **more flexible than hierarchy**
  - Ex: no existence dependence
- **Record-at-a-time** data manipulation language

CODASYL Example

- Figure 5 from “What goes around comes around”



CODASYL Limitations

- **No physical data independence**
  - Application programs break if organization changes
- **No logical data independence**
  - Application programs break if organization changes
- Very **complex**
- Programs must “**navigate** the hyperspace”
- Load and recover as **one gigantic object**


The Programmer as Navigator

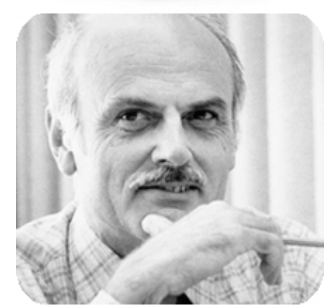
by Charles W. Bachman

Outline

- Different types of data
- Early data models
  - IMS
  - CODASYL
- Physical and logical independence in the relational model
- Data models that followed the relational model
- NoSQL data models

Relational Model Overview

- Proposed by Ted Codd in 1970
- Motivation: better logical and physical data independence
- Overview
  - Store data in a **simple data structure** (table)
    - Facilitates logical data independence
    - Flexible enough to represent almost anything
  - Access data through **set-at-a-time** language
    - Facilitates physical data independence
  - **No need for physical storage proposal**



Relational Database: A Practical Foundation for Productivity

Physical Independence

- Definition: **Applications are insulated from changes in physical storage details**

- Early models (IMS and CODASYL): No

- Relational model: Yes
  - Yes through set-at-a-time language: algebra or calculus
  - No specification of what storage looks like
  - Administrator can optimize physical layout

Logical Independence

- Definition: **Applications are insulated from changes to logical structure of the data**

- Early models
  - IMS: some logical independence
  - CODASYL: no logical independence

- Relational model
  - Yes through views

Views

- **View is a relation**
- But rows not explicitly stored in the database
- Instead
- **Computed as needed from a view definition**

Example with SQL

Relations

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)
```

```
CREATE VIEW Big_Parts
AS
SELECT * FROM Part WHERE psize > 10;
```

Example 2 with SQL

```
CREATE VIEW Supply_Part2 (name,no)
AS
SELECT R.sname, R.sno
FROM Supplier R, Supply S
WHERE R.sno = S.sno AND S.pno=2;
```

Queries Over Views

```
SELECT * from Big_Parts
WHERE pcolor='blue';

SELECT name
FROM Supply_Part2
WHERE no=1;
```

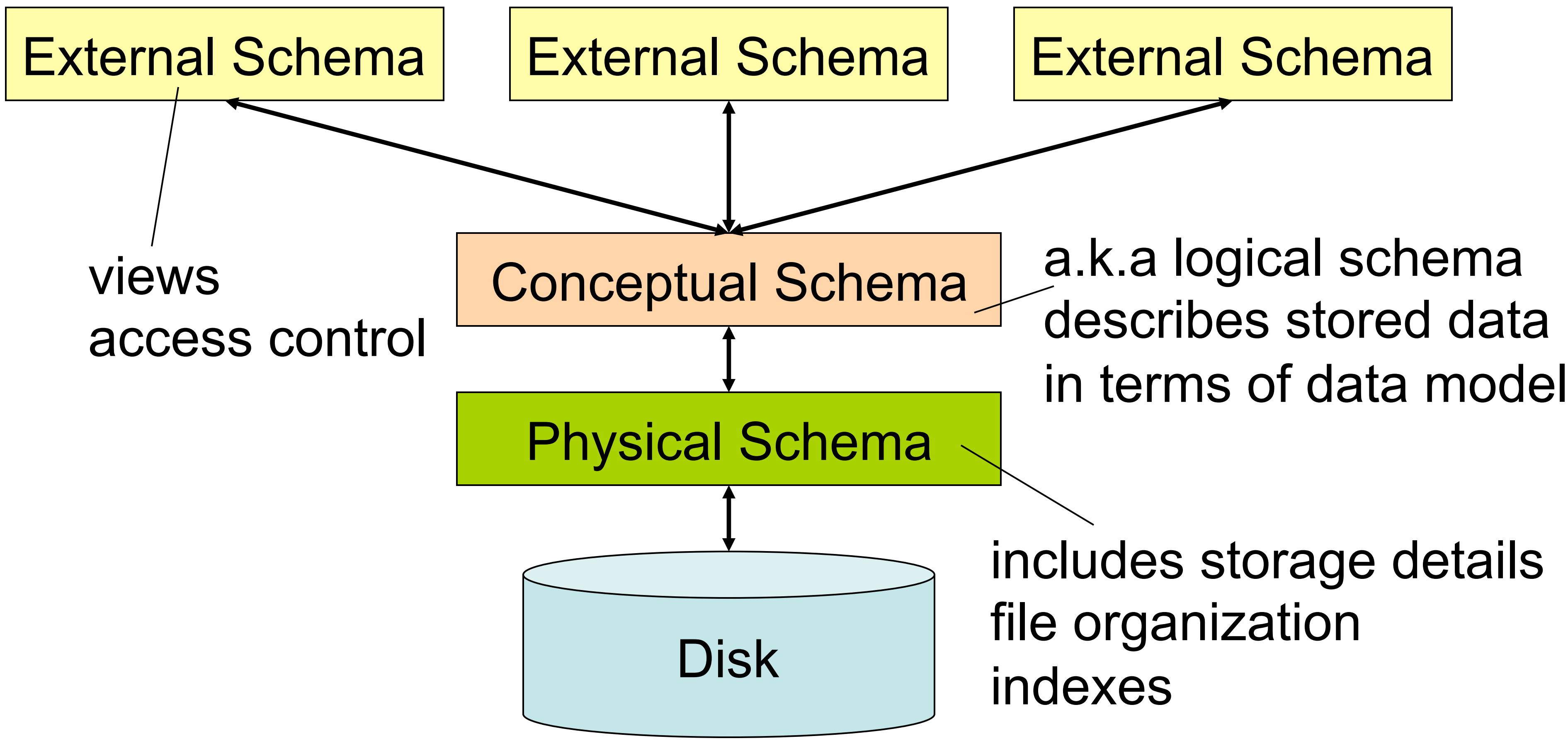
Updating Through Views

- **Updatable views** (SQL-92)
  - Defined on single base relation
  - No aggregation in definition
  - Inserts have NULL values for missing fields
  - Better if view definition includes primary key

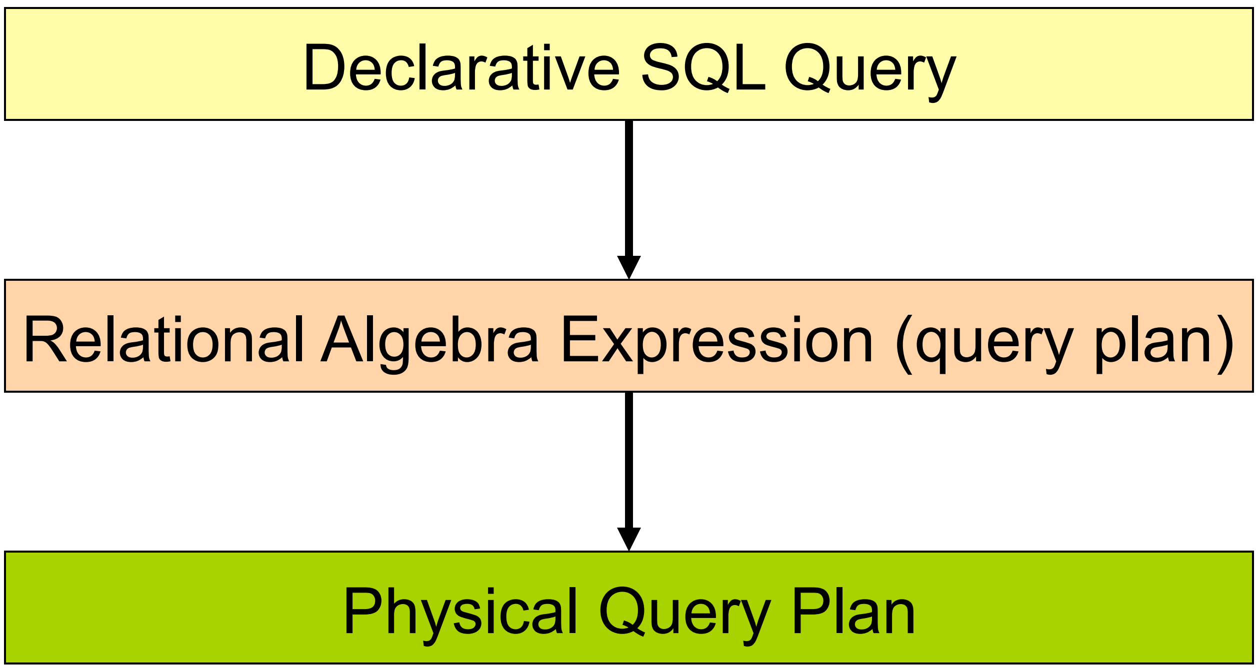
- Updatable views (SQL-99)
  - May be defined on multiple tables

- **Messy issue in general**

Levels of Abstraction



Query Translations



User or application

Optimizer

Great Debate

- Pro relational
  - What were the arguments?
- Against relational
  - What were the arguments?
- How was it settled?

Great Debate

- Pro relational
  - CODASYL is too complex
  - CODASYL does not provide sufficient data independence
  - Record-at-a-time languages are too hard to optimize
  - Trees/networks not flexible enough to represent common cases
- Against relational
  - COBOL programmers cannot understand relational languages
  - Impossible to represent the relational model efficiently
- Ultimately settled by the market place

Outline

- Different types of data
- Early data models
  - IMS
  - CODASYL
- Physical and logical independence in the relational model
- Data models that followed the relational model
- NoSQL data models

Other Data Models

- Entity-Relationship: 1970's
  - Successful in logical database design (last lecture)
- Extended Relational: 1980's
- Semantic: late 1970's and 1980's
- Object-oriented: late 1980's and early 1990's
  - Address impedance mismatch: relational dbs ←→ OO languages
  - Interesting but ultimately failed (several reasons, see references)
- Object-relational: late 1980's and early 1990's
  - User-defined types, ops, functions, and access methods
- Semi-structured: late 1990's to the present

Semistructured vs Relational

- Relational data model
  - Rigid flat structure (tables)
  - Schema must be fixed in advanced
  - Binary representation: good for performance, bad for exchange
  - Query language based on Relational Calculus

- Semistructured data model / XML
  - Flexible, nested structure (trees)
  - Does not require predefined schema ("self describing”)
  - Text representation: good for exchange, bad for performance
  - Query language borrows from automata theory

XML Syntax

```
<bibliography>
     <book>    <title> Foundations… </title>
               <author> Abiteboul </author>
               <author> Hull </author>
               <author> Vianu </author>
               <publisher> Addison Wesley </publisher>
               <year> 1995 </year>
     </book>
     …
</bibliography>
```

XML describes the content

Document Type Definitions (DTD)

- An XML document may have a DTD
- XML document:

  Well-formed = if tags are correctly closed

  Valid = if it has a DTD and conforms to it
- Validation is useful in data exchange
  - Use http://validator.w3.org/check to validate

Superseded by XML Schema (Book Sec. 11.4)

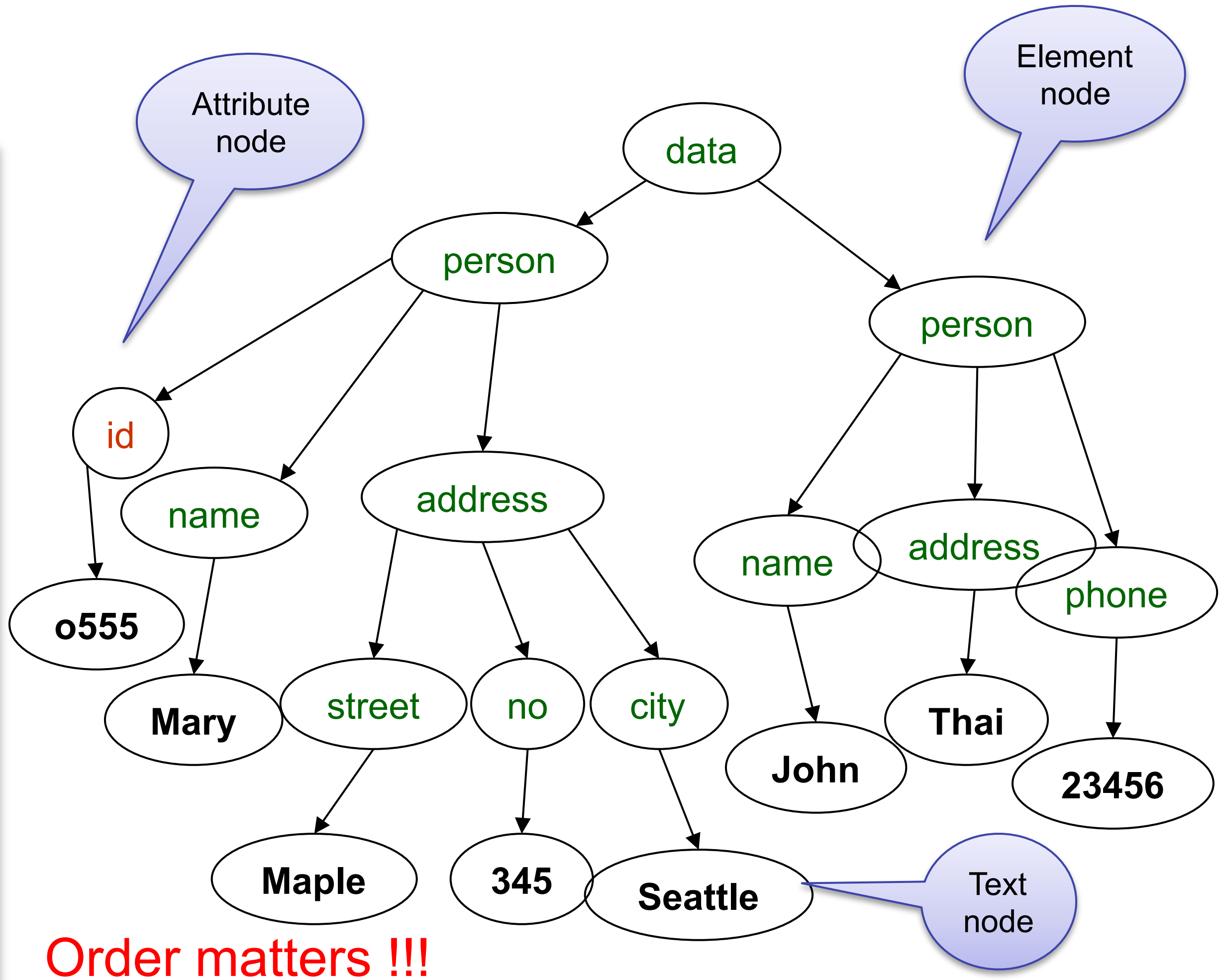
- Very complex: DTDs still used widely

Example DTD

```
<!DOCTYPE company [
  <!ELEMENT company   ((person|product)*)>
  <!ELEMENT person  (ssn, name, office, phone?)>
  <!ELEMENT ssn        (#PCDATA)>
  <!ELEMENT name      (#PCDATA)>
  <!ELEMENT office     (#PCDATA)>
  <!ELEMENT phone    (#PCDATA)>
  <!ELEMENT product  (pid, name, description?)>
  <!ELEMENT pid    (#PCDATA)>
  <!ELEMENT description    (#PCDATA)>
]>
```

XML Semantics: a Tree !

```
<data>
  <person id="o555" >
    <name> Mary </name>
    <address>
      <street>Maple</street>
      <no> 345 </no>
      <city> Seattle </city>
    </address>
  </person>
  <person>
    <name> John </name>
    <address>Thailand
    </address>
    <phone>23456</phone>
  </person>
</data>
```



Order matters !!!

Query XML with XQuery

FLWR ("Flower") Expressions

```
FOR $b IN doc("bib.xml")/bib
LET $a := distinct-values($b/book/author/text())
FOR $x IN $a
RETURN
  <answer>
    <author> $x </author>
    { FOR $y IN $b/book[author/text()=$x]/title
      RETURN $y }
  </answer>
```

SQL and XQuery Side-by-side

Product(pid, name, maker, price)

Find all product names, prices, sort by price

```
SELECT x.name,
       x.price
FROM Product x
ORDER BY x.price
```

SQL

```
FOR $x in doc("db.xml")/db/Product/row
ORDER BY $x/price/text()
RETURN <answer>
          { $x/name,  $x/price }
       </answer>
```

XQuery

Beyond XML: JSON

- JSON stands for “**J**ava**S**cript **O**bject **N**otation”
  - Lightweight text-data interchange format
  - Language independent
  - “Self-describing" and easy to understand

- JSON is quickly replacing XML for
  - Data interchange
  - Representing and storing semi-structure data

JSON

```
Example from: http://www.jsonexample.com/
myObject = {
   "first": "John",
   "last": "Doe",
   "salary": 70000,
   "registered": true,
   "interests": [ "Reading", "Biking", "Hacking" ]
}
```

Query language: JSONiq http://www.jsoniq.org/

Google Protocol Buffers

- Extensible way of serializing structured data
  - Language-neutral
  - Platform-neutral
- Used in communications protocols, data storage, etc.
- How it works
  - Developer specifies the schema in .proto file
  - Proto file gets compiled to classes that read/write the data
    - Compiler is language specific

https://developers.google.com/protocol-buffers/docs/overview

Google Protocol Buffers Example

```
From: https://developers.google.com/protocol-buffers/
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
  enum PhoneType { MOBILE = 0; HOME = 1; WORK = 2; }
  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }
  repeated PhoneNumber phone = 4;
}
```

Summary of Old Data Models

- Relational data model wins for data representation because of data independence
- E/R diagrams used in schema design
- Semistructured data (XML, JSON, Protocol Buffer) used in data exchange

Outline

- Different types of data
- Early data models
  - IMS
  - CODASYL
- Physical and logical independence in the relational model
- Data models that followed the relational model
- NoSQL data models

Cattell, SIGMOD Record 2010

Different Types of NoSQL

Taxonomy based on data models:

- Key-value stores
  - e.g., Project Voldemort, Memcached
- Extensible Record Stores
  - e.g., HBase, Cassandra, PNUTS
- Document stores
  - e.g., SimpleDB, CouchDB, MongoDB

NoSQL Data Models

- Key-value = each data item is a (key, value) pair
- Extensible record = families of attributes have a schema, but new attributes may be added
  - Hybrid between a tuple and a document
  - Families of attributes are defined in a schema
  - New attributes can be added (within a family) on per-record basis
  - Attributes may be list-valued
- Document = nested values, extensible records (think XML, JSON, attribute-value pairs)
  - **Values can be nested documents** or **lists** as well as **scalar** values
  - Attribute names are **dynamically defined for each doc at runtime**
  - Attributes are not defined in a global schema

NoSQL Data Models

- We will discuss NoSQL systems later in the quarter and will come back to their data models and operations

Conclusion

- Data independence is desirable
  - Both physical and logical
  - Early data models provided very limited data independence
  - Relational model facilitates data independence
    - Set-at-a-time languages facilitate phys. indep. [more next lecture]
    - Simple data models facilitate logical indep. [more next lecture]
- Flat models are also simpler, more flexible
- User should specify what they want not how to get it
  - Query optimizer does better job than human
- New data model proposals must
  - Solve a “major pain” or provide significant performance gains