CSE 544
Data Models

Lecture #3

Announcements

- Project
  - – Form groups by Friday
  - – Start thinking about a topic (see new additions to the topic list)
- Next paper review: due on Monday
- Homework 1: due the following Monday
- Makeup lecture:
  - – Tomorrow (Friday), 11am, CSE 403

Data Models

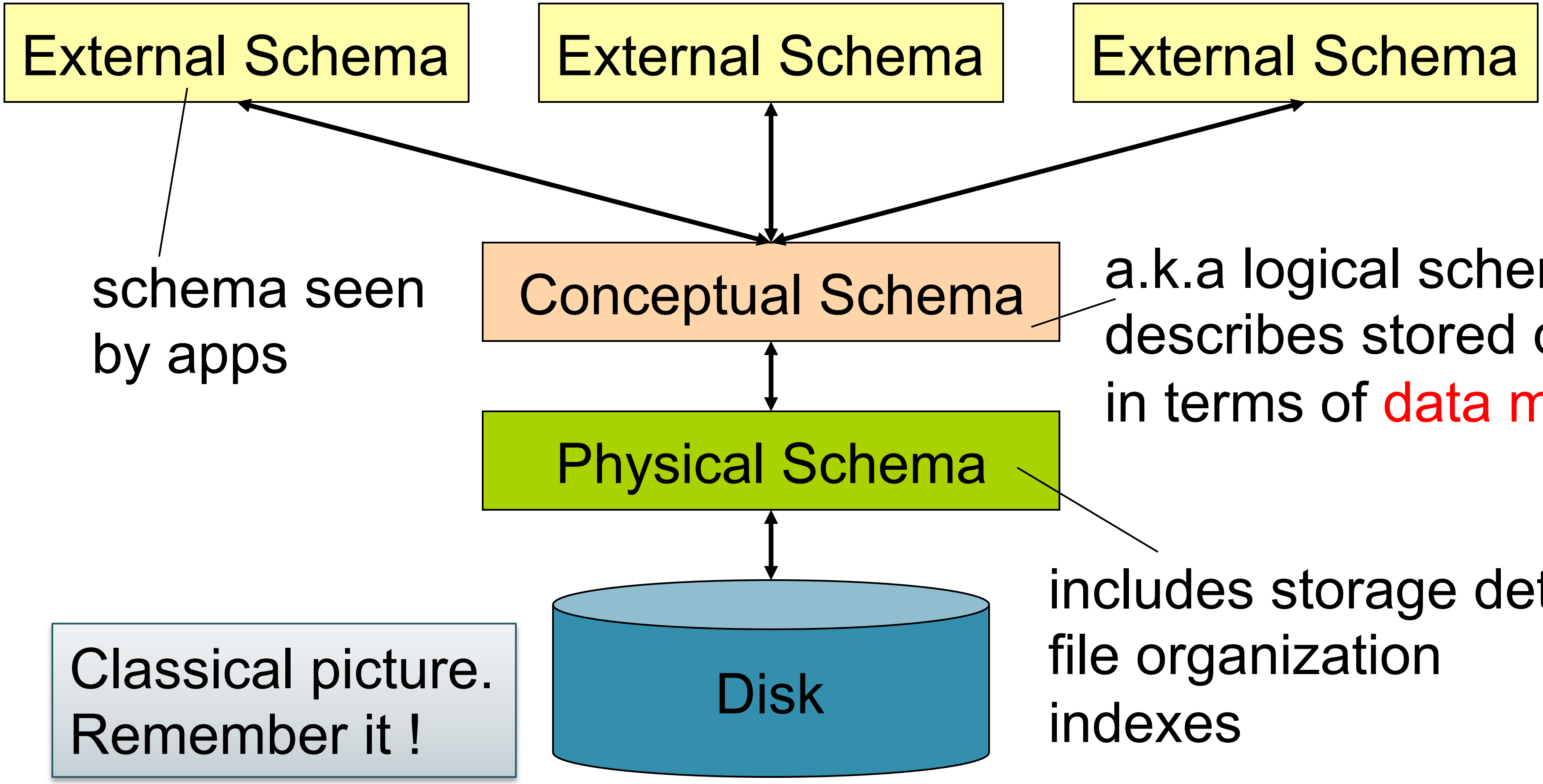
- M. Stonebraker and J. Hellerstein. What Goes Around Comes Around. In "Readings in Database Systems" (aka the Red Book). 4th ed.

“Data Model”

- Apps need to model real-world data
  - Typically includes entities and relationships between them
  - Entities: e.g. tudents, courses, products, clients
  - Relationships: e.g. course registrations, product purchases

- 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



External Schema

External Schema

External Schema

schema seen by apps

Conceptual Schema

a.k.a logical schema
describes stored data
in terms of data model

Physical Schema

includes storage details
file organization
indexes

Disk

Classical picture.
Remember it !

Outline

- Different types of data
- Early data models
  - IMS
  - CODASYL
- Relational model
- Other data models: E/R Diagrams, XML

Different Types of Data

- **Structured data**
  - – What is this ? Examples ?
- **Semistructured data**
  - – What is this ?
  - – Examples ?
- **Unstructured data**
  - – What is this ? Examples ?

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

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

- See Figure 2 in paper “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 with 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** (+)
- Network model is **more flexible than hierarchy(+)**
  - Ex: no existence dependence
- 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
- **Record-at-a-time** data manipulation language (-)

CODASYL Example

- See Figure 5 in paper “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**

Relational Model Overview

- Proposed by Ted Codd in 1970
- Motivation: better logical and physical data independence

Relational Model Overview

- Defines logical schema only
  - No physical schema

- Set-at-a-time query language

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

Great Debate

- Pro relational
  - What where the arguments ?
- Against relational
  - What where 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
  - CODASYL can represent tables

- Ultimately settled by the market place

Other Data Models

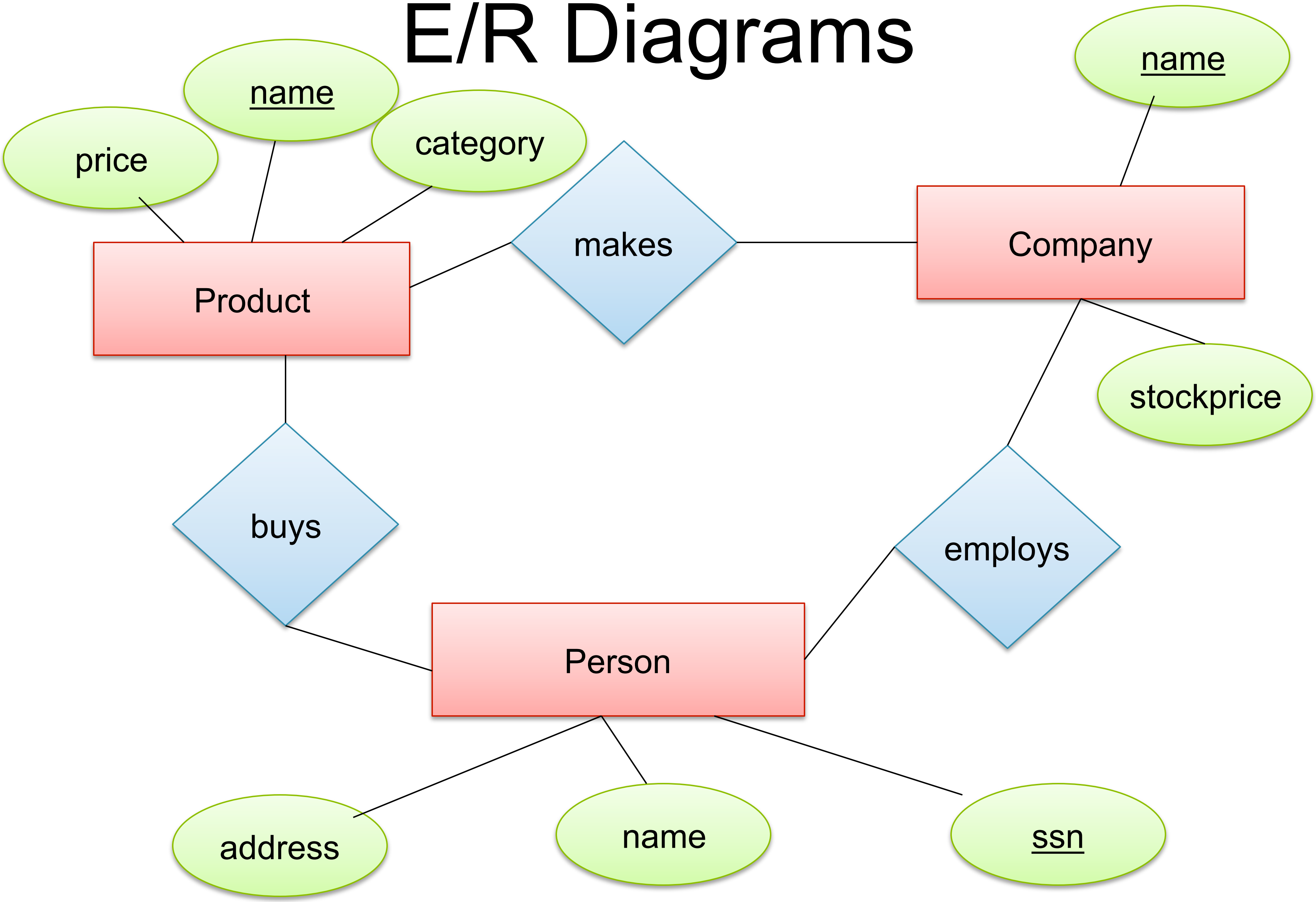
- Entity-Relationship: 1970’s
  - Successful in logical database design (you’ll use it in hw1)
- Extended Relational: 1980’s
- Semantic: late 1970’s and 1980’s
- Object-oriented: late 1980’s and early 1990’s
  - Address impedance mismatch: relational dbs ↔ OO languages
  - Interesting but ultimately failed (several reasons, see paper)
- 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

E/R Diagrams

Used today in *conceptual design*

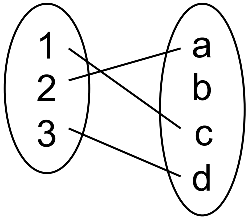
- Define the overall structure of the database; describe the entity sets, the attributes, and the relationships

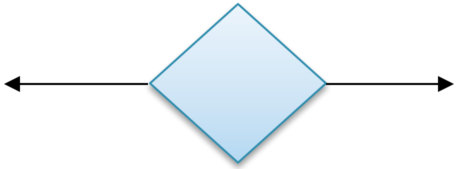
E/R Diagrams


name
price
category
makes
name
Company
Product
stockprice
buys
employs
Person
address
name
ssn

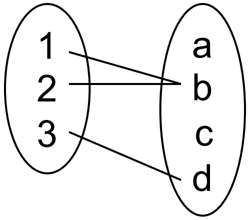
Multiplicity of E/R Relations

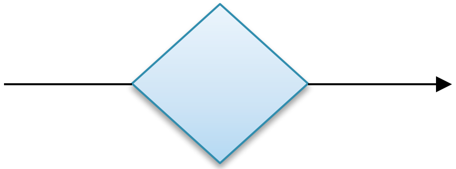
- one-one:



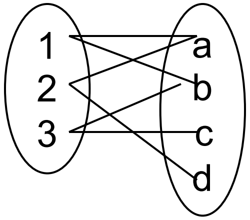


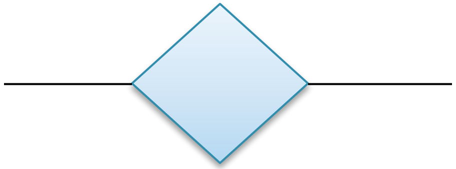
- many-one

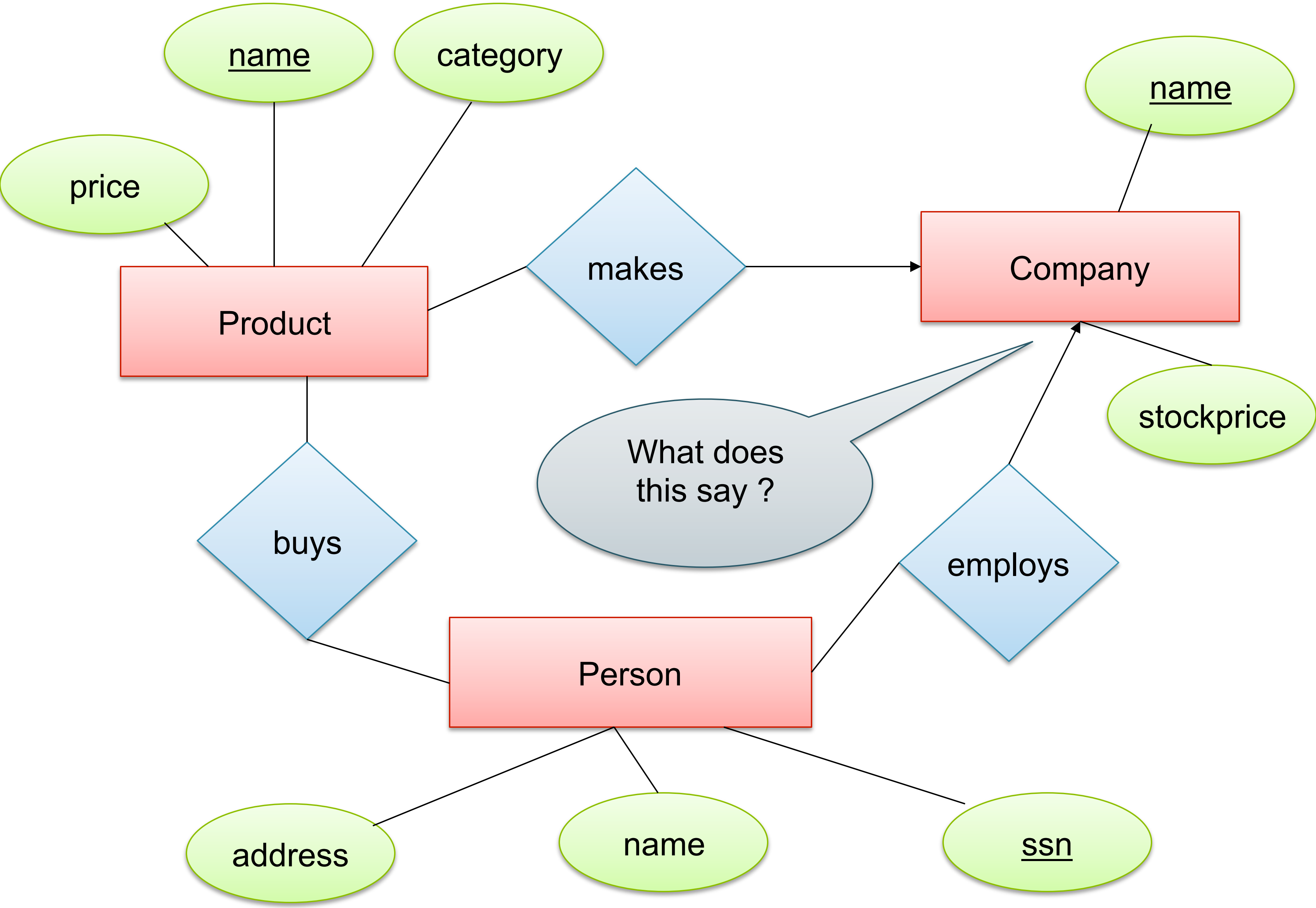




- many-many

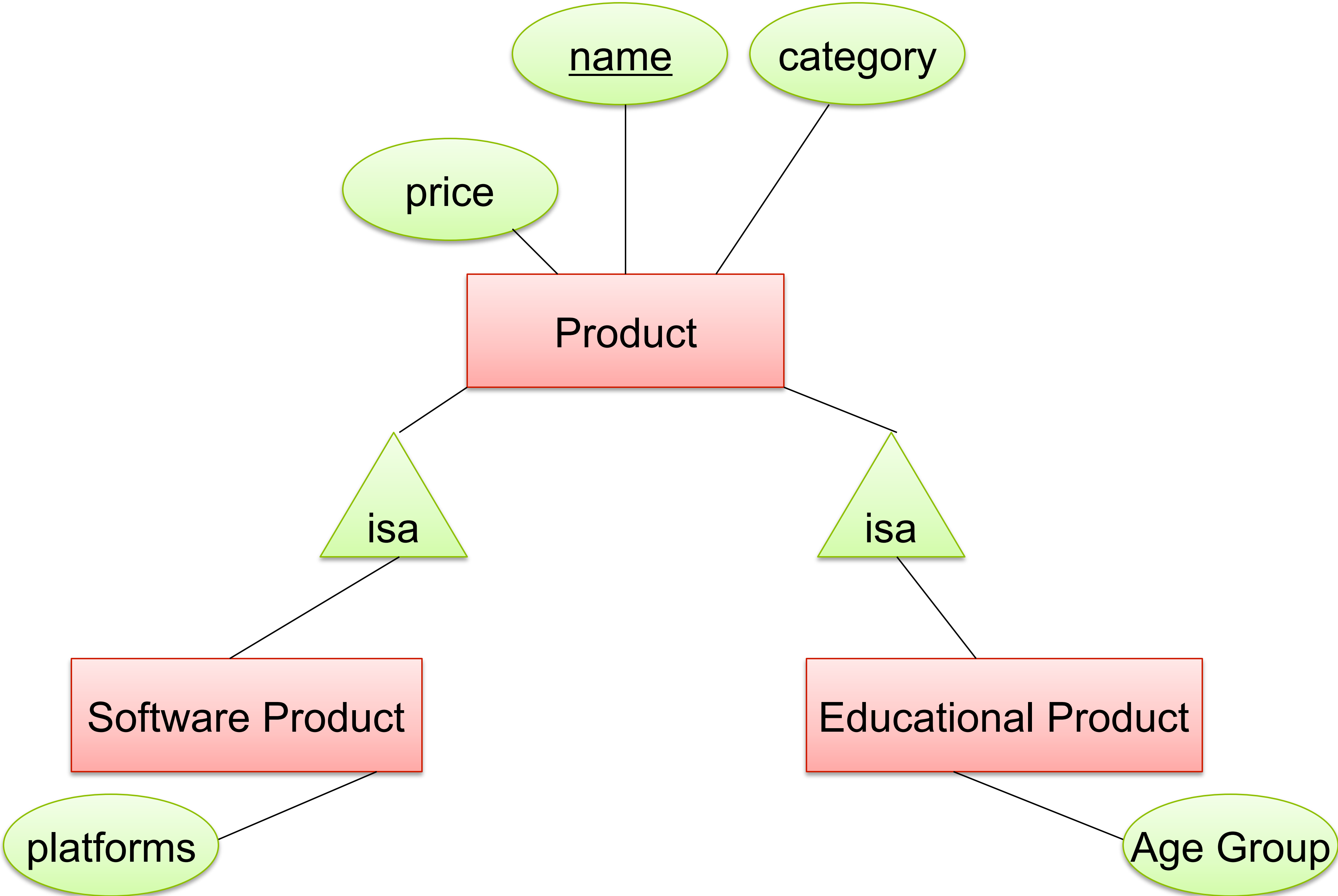




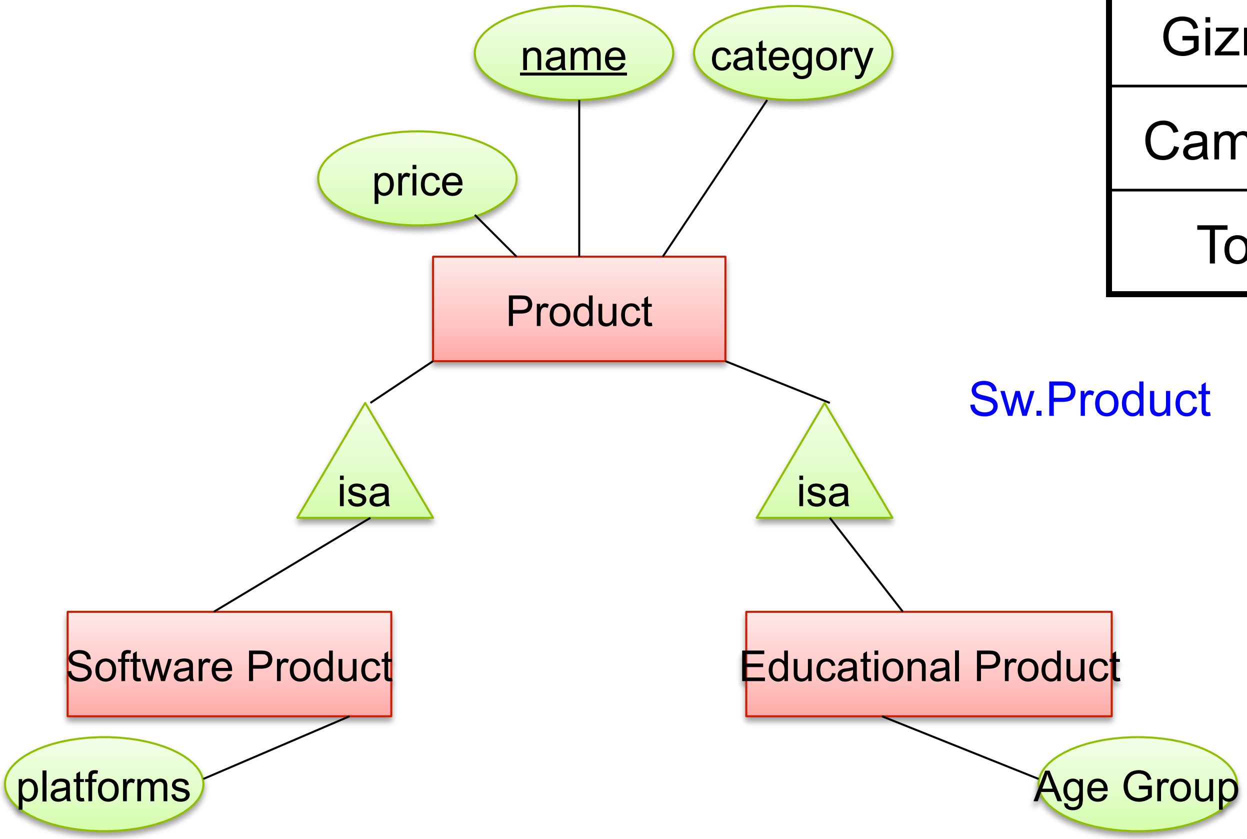


What does this say ?

Subclasses


name
category
price
Product
isa
isa
Software Product
Educational Product
platforms
Age Group

Subclasses to Relations



Product

<u>Name</u>	Price	Category
Gizmo	99	gadget
Camera	49	photo
Toy	39	gadget

Sw.Product

<u>Name</u>	platforms
Gizmo	unix

Ed.Product

<u>Name</u>	Age Group
Gizmo	todler
Toy	retired

Other ways to convert are possible

Semistructured Data and XML

- Two independent developments:
- Academia:
  - Wanted a flexible data model
  - *Schema first*
  - E.g., make it easy for data integration
- W3C standards committee
  - Created XML as an alternative to HTML to define *content* rather than *presentation*

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 Terminology

- Tags: book, title, author, …
- Start tag: <book>, end tag: </book>
- Elements: <book>…</book>,<author>…</author>
- Elements are nested
- Empty element: <red></red> abbrv. <red/>
- An XML document: single *root element*

Well formed XML document

- Has matching tags
- A short header
- And a root element

Well-Formed XML

```
<? xml version="1.0" encoding="utf-8" standalone="yes" ?>
<SomeTag>
     …
</SomeTag>
```

Parsing and processing XML Documents:

- DOM = Document Object Model = main memory
- SAX = Simple API for XML = event driven = we use it in HW1

More XML: Attributes

```
<book price = “55” currency = “USD”>
   <title> Foundations of Databases </title>
   <author> Abiteboul </author>
    …
   <year> 1995 </year>
</book>
```

Attributes v.s. Elements

```
<book price = “55” currency = “USD”>
   <title> Foundations of DBs </title>
   <author> Abiteboul </author>
    …
   <year> 1995 </year>
</book>
```

```
<book>
  <title> Foundations of DBs </title>
   <author> Abiteboul </author>
    …
   <year> 1995 </year>
   <price> 55 </price>
   <currency> USD </currency>
</book>
```

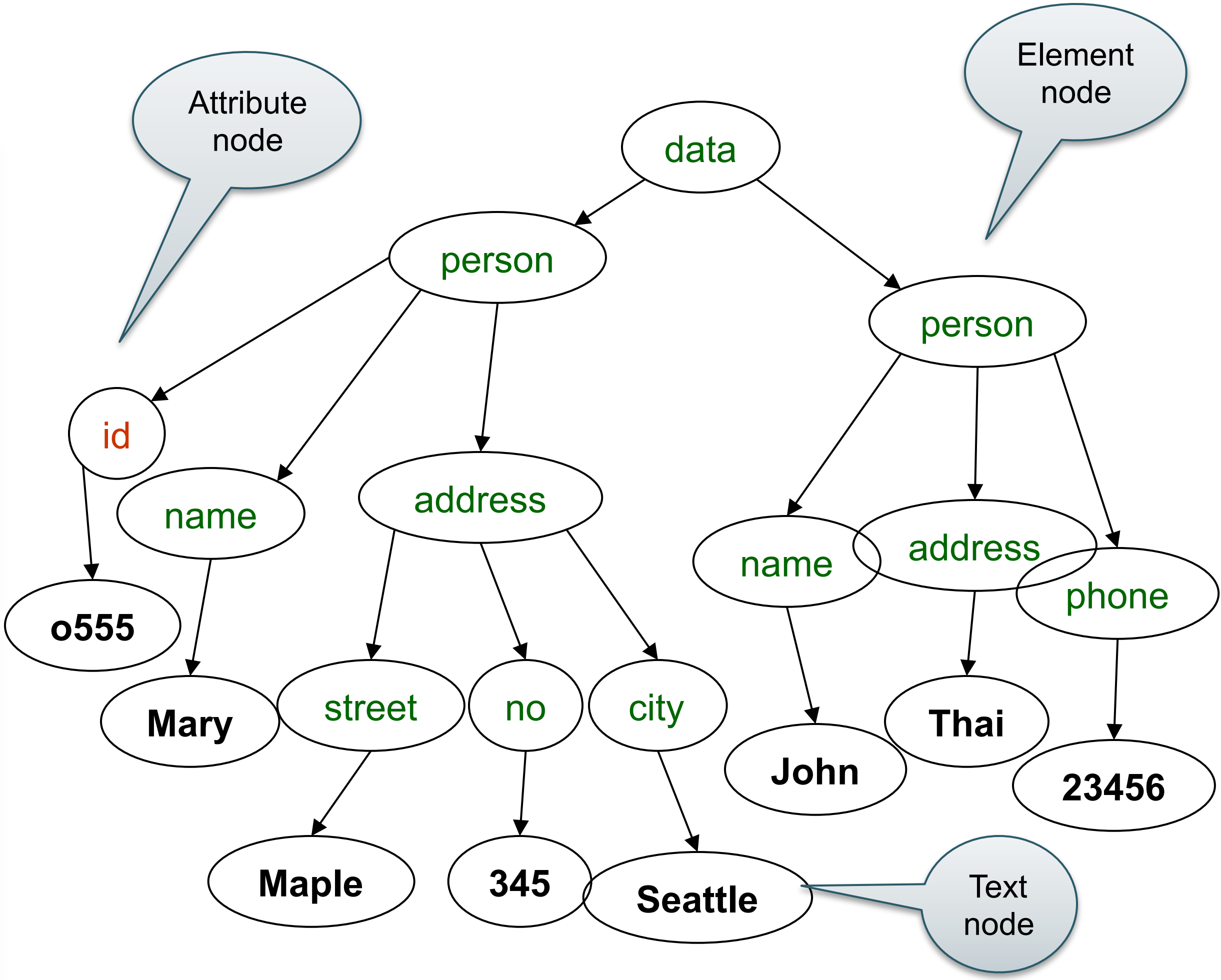
Attributes are alternative ways to represent data

Comparison

Elements	**Attributes**
Ordered	Unordered
May be repeated	Must be unique
May be nested	Must be atomic

XML Semantics: a Tree !
DOM = Document Object Model

```
<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 !!!

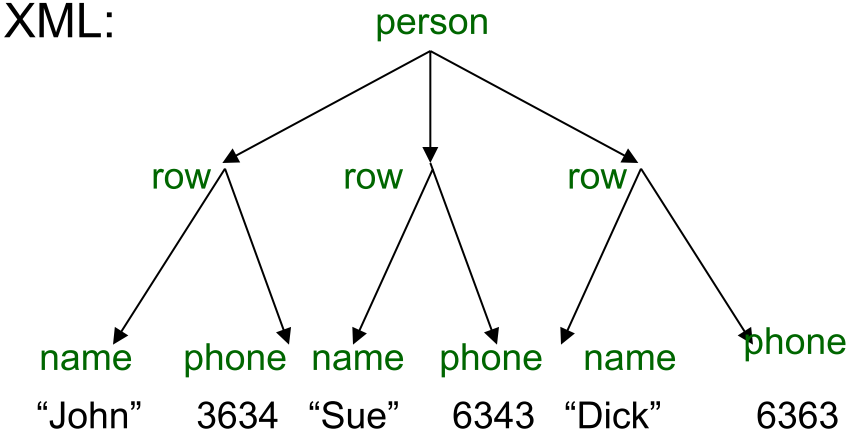
XML Data

- XML is self-describing
- Schema elements become part of the data
  - Relational schema: person(name,phone)
  - In XML <person>, <name>, <phone> are part of the data, and are repeated many times
- Consequence: XML is much more flexible
- XML = semistructured data

Mapping Relational Data to XML Data

The canonical mapping:

XML:



Person

Name	Phone
John	3634
Sue	6343
Dick	6363

```
<person>
      <row> <name>John</name>
                 <phone> 3634</phone></row>
       <row> <name>Sue</name>
                  <phone> 6343</phone></row>
       <row> <name>Dick</name>
                  <phone> 6363</phone></row>
</person>
```

Mapping Relational Data to XML Data

XML

Application specific mapping

Person

Name	Phone
John	3634
Sue	6343

Orders

PersonName	Date	Product
John	2002	Gizmo
John	2004	Gadget
Sue	2002	Gadget

```
<people>
  <person>
      <name> John </name>
      <phone> 3634 </phone>
      <order>  <date> 2002 </date>
               <product> Gizmo </product>
       </order>
       <order> <date> 2004 </date>
               <product> Gadget </product>
       </order>
  </person>
 <person>
      <name> Sue </name>
      <phone> 6343 </phone>
      <order>  <date> 2004 </date>
               <product> Gadget </product>
       </order>
  </person>
</people>
```

XML=Semi-structured Data (1/3)

- Missing attributes:

```
<person>   <name> John</name>
           <phone>1234</phone>
 </person>

<person>  <name>Joe</name>
</person>
```

no phone !

- Could represent in a table with nulls

name	phone
John	1234
Joe	-

XML=Semi-structured Data (2/3)

- Repeated attributes

```
<person> <name> Mary</name>
            <phone>2345</phone>
            <phone>3456</phone>
</person>
```

Two phones !

- Impossible in tables:

name	phone	
Mary	2345	3456

???

XML=Semi-structured Data (3/3)

- Attributes with different types in different objects

```
<person> <name>  <first> John </first>
                  <last> Smith </last>
           </name>
           <phone>1234</phone>
</person>
```

Structured name !

- Nested collections
- Heterogeneous collections:
  - <db> contains both <book>s and <publisher>s

Summary

- 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