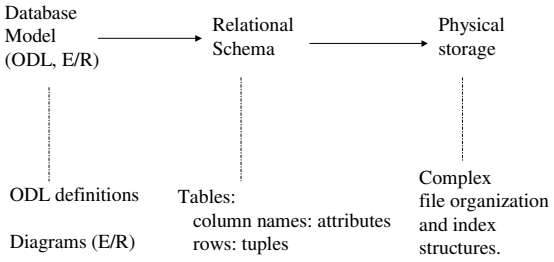
The Relational Data Model

Monday, October 2, 2000

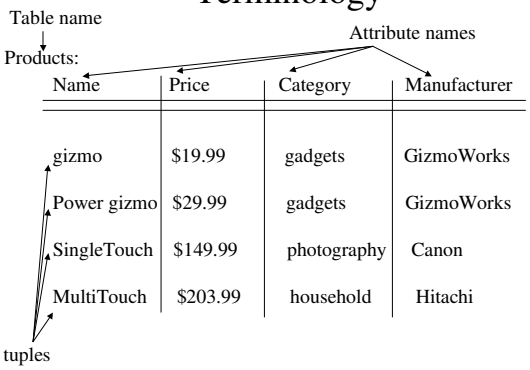
Outline

- The relational model (3.1)
- ODL to relational model (3.2)
- E/R to relational model (3.3)
- Subclasses to relational model (3.4)

The Relational Data Model



Terminology



Domains

- each attribute has a type
- must be atomic type (why ? see later)
- called *domain*
- examples:
  - Integer
  - String
  - Real
  - …

Schemas

- Relational Schema:
  - Relation name plus attribute names
  - E.g. Product(Name, Price, Category, Manufacturer)
  - In practice we add the domain for each attribute
- Database Schema
  - Set of relational schemas
  - E.g. Product(Name, Price, Category, Manufacturer)
    Vendor(Name, Address, Phone)

Instances

- An instance of a relational schema R(A1,…,Ak), is a relation with k attributes with values of corresponding domains
- An instance of a database schema R1(…), R2(…), …, Rn(…), consists of n relations, each an instance of the corresponding relational schema.

Example

Relational schema: Product(Name, Price, Category, Manufacturer)
Instance:

Name	Price	Category	Manufacturer
gizmo	$19.99	gadgets	GizmoWorks
Power gizmo	$29.99	gadgets	GizmoWorks
SingleTouch	$149.99	photography	Canon
MultiTouch	$203.99	household	Hitachi

Updates

The database maintains a current database state.

Updates to the data:

1) add a tuple
2) delete a tuple
3) modify an attribute in a tuple

Updates to the data happen very frequently.

Updates to the schema: relatively rare. Rather painful. **Why?**

Schemas and Instances

- Analogy with programming languages:
  - Schema = type
  - Instance = value
- Important distinction:
  - Database Schema = stable over long periods of time
  - Database Instance = changes constantly, as data is inserted/updated/deleted

Two Mathematical Definitions of Relations

- Relation as cartesian product
  - tuple = element of **string** x **int** x **string** x **string**
  - E.g. t = (gizmo, 19, gadgets, GizmoWorks)
  - Product is subset of **string** x **int** x **string** x **string**
  - Order in the tuple is important !
    - (gizmo, 19, gadgets, GizmoWorks)
    - (gizmo, 19 , GizmoWorks, gadgets)
  - No attributes

- Relation as a set of functions
  - Fix the set of attributes
    - A={name , price, category, manufacturer}
  - A tuple is a function t:A → Domains
  - Relation = set of tuples
  - E.g. {name → gizmo, price → 19, category → gadgets, manufacturer→gizmoWorks}
  - Order in a tuple is not important
  - Attribute names are important

Two Definitions of Relations

- We will switch back and forth between these two:
  - Relational schemas with attribute names
  - Positional tuples, without attribute names

From ODL to Relational Schema

Start simple: a class definition has only single valued attributes

Interface product{
float price;
string name;
Enum {telephony, gadgets, books} category}

Class becomes a relation, and every attribute becomes a relation attribute:

Product

Name	Price	Category
Gizmo	$19.99	gadgets

Adding Non atomic Attributes

Price is a record: {string currency, float amount}

Product

Name	Currency	Amount	Category
Gizmo	US$	19.99	gadgets
Power Gizmo	US$	29.99	gadgets

Set Attributes

Interface person{
string name;
integer SSN;
set <integer> PhoneNumber;}

One option: have a tuple for every value in the set:

Name	SSN	PhoneNumber
Fred	123-321-99	(201) 555-1234
Fred	123-321-99	(206) 572-4312
Joe	909-438-44	(908) 464-0028
Joe	909-438-44	(212) 555-4000

Disadvantages?

Modeling Collection Types

The problem becomes even more significant if a class has several attributes that are set types?
Question: how bad is the redundancy for *n* set type attributes, each with possibly up to *m* values?

Questions:
How can we model bags?

Lists?

Fixed length arrays?

Modeling Relationships

Interface Product {
attribute string name;

attribute float price;
relationship <Company> madeBy;
}
Interface Company {

attribute string name;
attribute float stock-price;
attribute string address;

}

How do we incorporate the relationship madeBy into the schema?

Option #1

Name	Price	made-by-name	made-by-stock-price	made-by-address
Gizmo	$19.99	gizmoWorks	0.0001$	Montezuma

What's wrong?

Hint

```
Interface Product {
        attribute  string name;

        attribute  float   price;
        relationship  Company  madeBy;
        }
Interface  Company {

         attribute   string  name;
         attribute   float stock-price;
         attribute   string  address;
         relationship  set <Product> makes;
  }
```

Better Solution

Product relation: (assume: name is a key for company)

Name	Price	made-by-name
Gizmo	$19.99	gizmoWorks

Company relation:

Name	Stock Price	Address
gizmoWorks	$0.00001	Montezuma

Additional Issues

1. What if there is no key?
2. What if the relationship is multi-valued?
3. How do we represent a relationship and its inverse?

Answer: that's what we have the book for ☺

From E/R Diagrams to Relational Schema

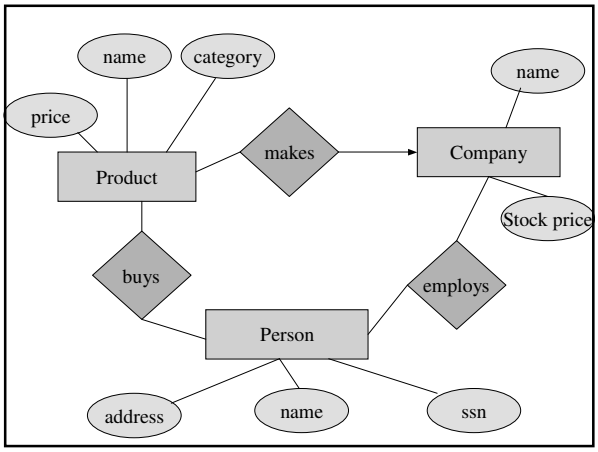
Easier than ODL (using a liberal interpretation of the word "easy")

- relationships are already independent entities
- only atomic types exist in the E/R model.

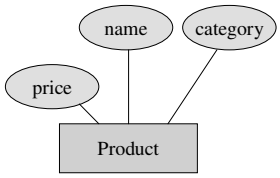
Entity sets ⟶ relations

Relationships ⟶ relations

Special care for weak entity sets.



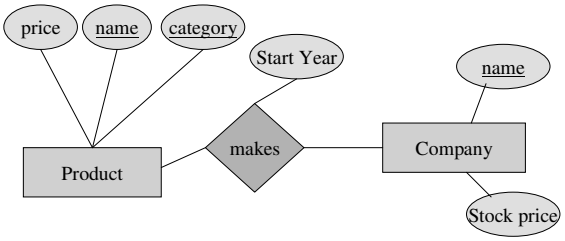
Entity Sets to Relations



Product:

Name	Category	Price
gizmo	gadgets	$19.99

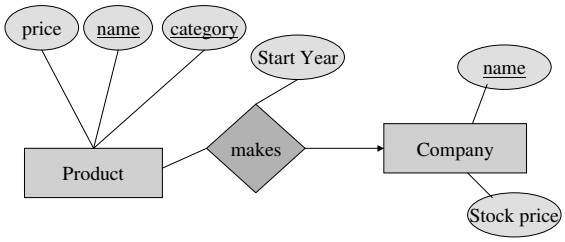
Relationships to Relations



Relation Makes (watch out for attribute name conflicts)

Product-name	Product-Category	Company-name	Starting-year
gizmo	gadgets	gizmoWorks	1963

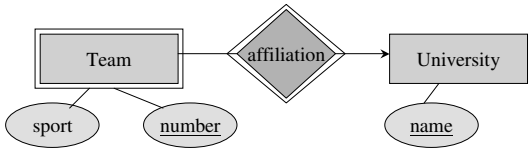
Many-one Relationships



No need for Makes. Just modify Product:

name	category	price	StartYear	companyName
gizmo	gadgets	19.99	1963	gizmoWorks

Handling Weak Entity Sets

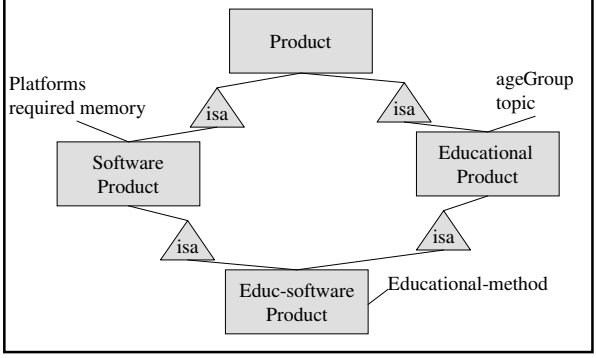


Relation Team:

Sport	Number	Affiliated University
mud wrestling	15	Montezuma State U.

- need all the attributes that contribute to the key of Team
- don't need a separate relation for Affiliation. (why ?)

Modeling Subclass Structure



Option #1: the ODL Approach

4 tables: each object can only belong to a single class
Product(name, price, category, manufacturer)

EducationalProduct(name, price, category, manufacturer, ageGroup, topic)

SoftwareProduct(name, price, category, manufacturer, platforms, requiredMemory)

EducationalSoftwareProduct(name, price, category, manufacturer, ageGroup, topic, platforms, requiredMemory)

All names are distinct

Option #2: the E/R Approach

Product(name, price, category, manufacturer)

EducationalProduct(name, ageGroup, topic)

SoftwareProduct(name, platforms, requiredMemory)

No need for a relation EducationalSoftwareProduct

Unless, it has a specialized attribute:
EducationalSoftwareProduct(name, educational-method)

Same name may appear in several relations

Option #3: The Null Value Approach

Have one table:

Product (name, price, manufacturer, age-group, topic, platforms, required-memory, educational-method)

Some values in the table will be NULL, meaning that the attribute not make sense for the specific product.

How many more meanings will NULL have??