Database Design

Sept. 27, 2000

Outline

- ODL - Object Definition Language (2.1)
- E/R - Entity relationship diagrams (2.2)
- Design Principles (2.3)

Database Design

- Why do we need it?
  - Agree on structure of the database before deciding on a particular implementation.
- Consider issues such as:
  - What entities to model
  - How entities are related
  - What constraints exist in the domain
  - How to achieve *good* designs

Database Design Formalisms

1. Object Definition Language (ODL):
   - Closer in spirit to object-oriented models
2. Entity/Relationship model (E/R):
   - More relational in nature.

- Both can be translated (semi-automatically) to relational schemas
- ODL to OO-schema: direct transformation (C++ or Smalltalk based system).

1. Object Definition Language

- ODL is part of ODMG
- superset of Corba's IDL
- Resembles C++ (and Smalltalk).

- Basic design paradigm in ODL:
  - Model ***objects*** and their ***properties.***
- For abstraction purposes:
  - Group objects into ***classes.***
- What qualifies as a ***good*** class?
  - Objects should have common properties.

ODL Class Declarations

Interface <name> {

attributes: <type> <name>;

relationships <range type> <name>;

methods

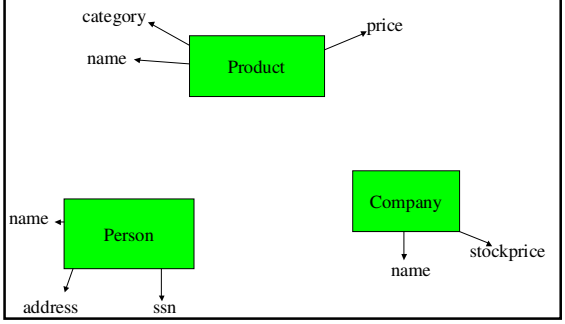
}

Method example:

float gpa(in: Student) **raises** (noGrades)

Arbitrary function can compute the value of gpa, based on a student object given as input.

ODL Example

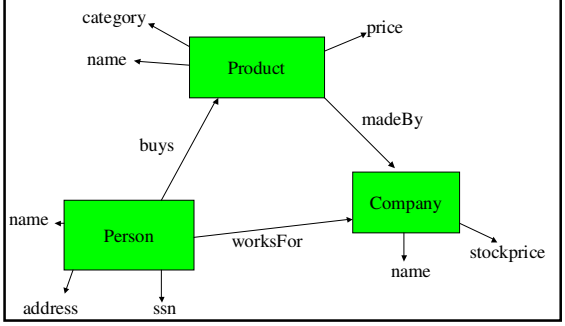


ODL Declarations

```
Interface Product {
      attribute  string name;
      attribute  float   price;
      attribute  enum Categories
                        {electronics, communications, sports …} category
      }
Interface Company {
      attribute   string  name;
      attribute   float    stockprice;
      }
Interface Person {
      attribute integer ssn;
      attribute string   name;
      attribute Struct Address {string street, string city} address; }
```

So far just simplified C++ with slightly different syntax

ODL Example Extended



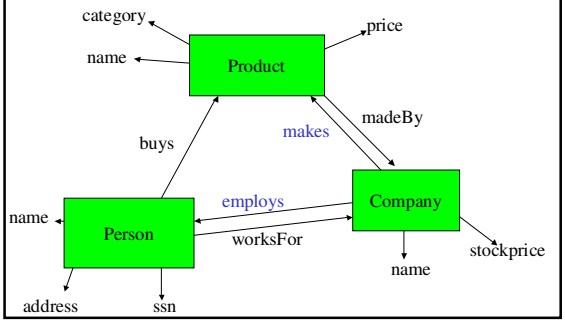
ODL Declarations, Extended

```
Interface Product {
      attribute  string name;
      attribute  float   price;
      attribute  enum Categories
                        {electronics, communications, sports …} category;
      relationship  <Company>  madeBy;
      }

Interface Person {
      attribute integer ssn;
      attribute string   name;
      attribute Struct Address {string street, string city} address;
      relationship  set <Product>  buys;
      relationship  set <Company>  worksFor;}
```

relationship corresponds somewhat to pointers in C++

ODL Example, Extended Again



ODL Declarations, Extended Again

```
Interface Company {
      attribute   string  name;
      attribute   float    stockprice;

       relationship  set <Product> makes
                     inverse  Product::madeBy;

       relationship   set <Person>  employs
                     inverse  Person::worksFor;
      }
```

Types in ODL

Basic types:

Atomic types (e.g., string, integer, …)
Interface types (e.g., Person, Product, Company)

Constructors:

collection types:
- Set: (1, 5, 6)
- Bag: (1, 1, 5, 6, 6)
- List: (1, 5, 6, 1, 6)
- Array: Integer[17]

structured types:
- Struct {string street, string city, integer zipcode}

Collection Types

- Sets:
  - order, number of occurrences don't matter
  - {4,7,9} = {7,9,7,4} = {9,4,7}
- Bags:
  - number of occurrences matter, order not:
  - {7,9,7,4}={7,7,9,4}, is different from {4,7,9}
- Lists:
  - order, number of occurrences matter:
  - [4,7,9] different from [9,4,7]

Allowable Types in ODL

For attributes: start with atomic or struct, and apply a collection type.

OK: string, set of integer, bag of Address.
Not OK: Product, set of set of integer.

For relationships:
start with interface type and apply a collection type.

OK: Product, set of Product, list of Person.

Not OK: **struct** {pname Product, cname Company}
set of bag of Product
integer

2. Entity / Relationship Diagrams

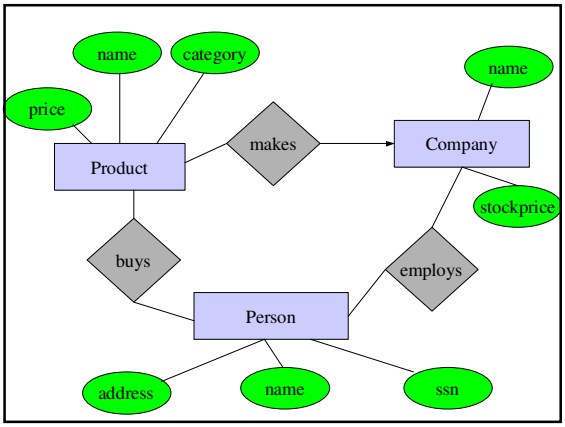
Objects ⟶ entities
Classes ⟶ entity sets

Attributes are like in ODL.

Relationships: like in ODL except

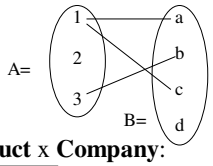
- not associated with classes (I.e., first class citizens)

- not necessarily binary



What is a Relation ?

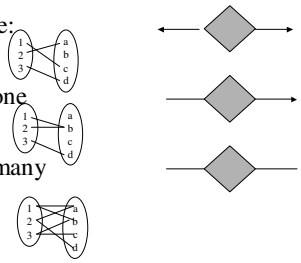
- A mathematical definition:
  - if A, B are sets, then a relation R is a subset of A x B
- A={1,2,3}, B={a,b,c,d},
  R = {(1,a), (1,c), (3,b)}



- **makes** is a subset of **Product** x **Company**:

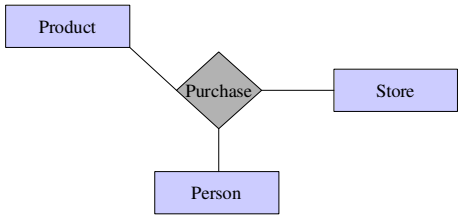
Multiplicity of E/R Relations

- one-one:
- many-one
- many-many



Multi-way Relationships

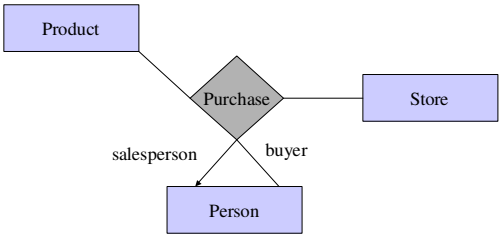
How do we model a purchase relationship between buyers, products and stores?



Can still model as a mathematical set (how ?)

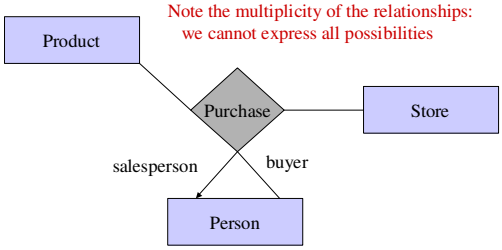
Roles in Relationships

What if we need an entity set twice in one relationship?

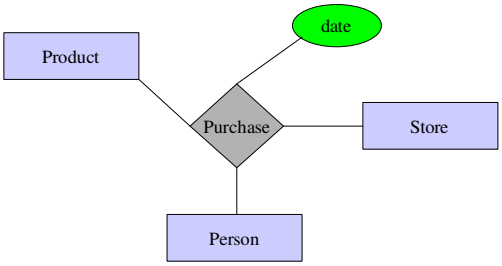


Roles in Relationships

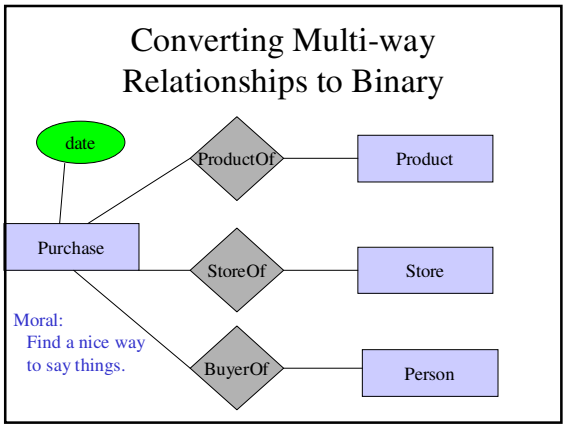
Note the multiplicity of the relationships: we cannot express all possibilities



Attributes on Relationships



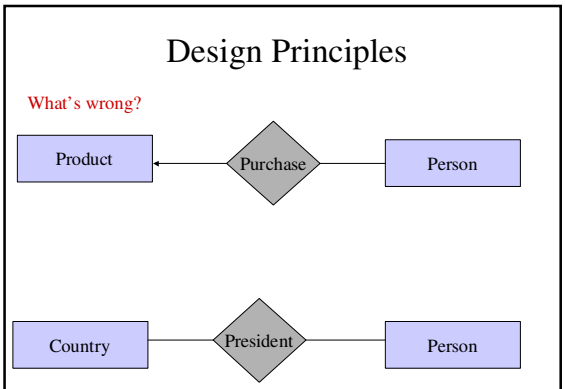
Converting Multi-way Relationships to Binary



Moral:
Find a nice way to say things.

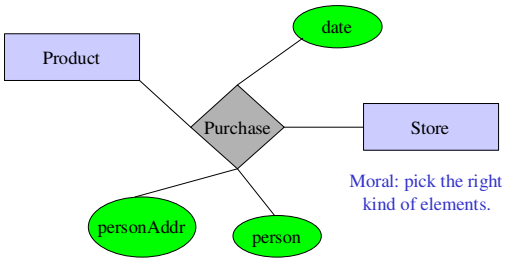
Design Principles

What's wrong?



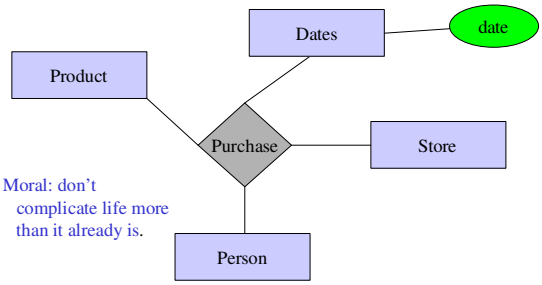
Moral: be faithful!

What's Wrong?



Moral: pick the right kind of elements.

What's Wrong?



Moral: don't complicate life more than it already is.