Lecture 08:
Wednesday, October 16, 2002

Outline
- Finish E/R diagrams (Chapter 2)
  - And E/R diagrams to relations (3.2, 3.3)
- The relational data model: 3.1
- Functional dependencies: 3.4

Difference between ODL and E/R inheritance
- ODL: classes are disjoint

Difference between ODL and E/R inheritance
- E/R: entity sets overlap

No need for multiple inheritance in E/R
- we have three entity sets, but four different kinds of objects
- Still needed if we want extra attributes
**Modeling Subclass Structure**

Platforms required memory

Software Product

*isa* 

Educational Product

*isa* 

Educ-software Product

*isa*

ageGroup topic

**Modeling UnionTypes With Subclasses**

Say: each piece of furniture is owned either by a person, or by a company

**Modeling Union Types with Subclasses**

Say: each piece of furniture is owned either by a person, or by a company

Solution 1. Acceptable, imperfect (What’s wrong ?)

Person

*isa* 

FurniturePiece

*isa* 

Company

ownedByPerson

ownedByPerson

**Modeling Union Types with Subclasses**

Solution 2: better, more laborious

FurniturePiece

Company

Owner

*isa* 

Person

ownedBy

**Constraints in E/R Diagrams**

Finding constraints is part of the modeling process. Commonly used constraints:

- **Keys**: social security number uniquely identifies a person.
- **Single-value constraints**: a person can have only one father.
- **Referential integrity constraints**: if you work for a company, it must exist in the database.
- **Other constraints**: peoples’ ages are between 0 and 150.

**Keys in E/R Diagrams**

Underline:

Product

name

category

price

No formal way to specify multiple keys in E/R diagrams

Person

name

address

Fun
Single Value Constraints

Referential Integrity Constraints

Other Constraints

Weak Entity Sets

Handling Weak Entity Sets

The Relational Data Model

What does this mean?

Entity sets are weak when their key comes from other classes to which they are related.

Convert to a relational schema (in class)
### Terminology

<table>
<thead>
<tr>
<th>Products:</th>
<th>Attribute names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Price</td>
</tr>
<tr>
<td>gizmo</td>
<td>$19.99</td>
</tr>
<tr>
<td>Power gizmo</td>
<td>$29.99</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
</tr>
</tbody>
</table>

Tables or rows or records

### 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),
  Company(Name, Address, Phone),
  . . . . . .`

This is all mathematics, not to be confused with SQL tables!

### Instances

- **Relational schema** = $R(A_1, \ldots, A_k)$:
  - **Instance** = relation with k attributes (of “type” $R$)
    - values of corresponding domains

- **Database schema** = $R_1(\ldots), R_2(\ldots), \ldots, R_n(\ldots)$
  - **Instance** = n relations, of types $R_1, R_2, \ldots, R_n$

### Example

**Relational schema:** `Product(Name, Price, Category, Manufacturer)`

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>gizmo</td>
<td>$19.99</td>
<td>gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Power gizmo</td>
<td>$29.99</td>
<td>gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

### First Normal Form (1NF)

- A database schema is in First Normal Form if all tables are flat

### Functional Dependencies

- A form of constraint
  - hence, part of the schema
- Finding them is part of the database design
- Also used in normalizing the relations
**Functional Dependencies**

**Definition:**

If two tuples agree on the attributes $A_1, A_2, \ldots, A_n$ then they must also agree on the attributes $B_1, B_2, \ldots, B_m$.

Formally: $A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$

---

**Examples**

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E1847</td>
<td>John</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

- EmpID $\rightarrow$ Name, Phone, Position
- Position $\rightarrow$ Phone
- but Phone $\rightarrow$ Position

---

**In General**

- To check $A \rightarrow B$, erase all other columns

<table>
<thead>
<tr>
<th>...</th>
<th>A</th>
<th>...</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td></td>
<td>Y1</td>
<td></td>
</tr>
<tr>
<td>X2</td>
<td></td>
<td>Y2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

- check if the remaining relation is many-one (called functional in mathematics)

*Note: this is the mathematical definition of a function. Book is wrong.*

---

**Typical Examples of FDs**

- **Product:** name $\rightarrow$ price, manufacturer
- **Person:** ssn $\rightarrow$ name, age
- **Company:** name $\rightarrow$ stockprice, president

---

**In Class: Find All FDs**

<table>
<thead>
<tr>
<th>Student</th>
<th>Dept</th>
<th>Course</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>CSE</td>
<td>C++</td>
<td>020</td>
</tr>
<tr>
<td>Bob</td>
<td>CSE</td>
<td>C++</td>
<td>020</td>
</tr>
<tr>
<td>Alice</td>
<td>EE</td>
<td>HW</td>
<td>040</td>
</tr>
<tr>
<td>Carol</td>
<td>CSE</td>
<td>DB</td>
<td>045</td>
</tr>
<tr>
<td>Dan</td>
<td>CSE</td>
<td>Java</td>
<td>050</td>
</tr>
<tr>
<td>Elsa</td>
<td>CSE</td>
<td>DB</td>
<td>045</td>
</tr>
<tr>
<td>Frank</td>
<td>EE</td>
<td>Circuits</td>
<td>020</td>
</tr>
</tbody>
</table>

*Do all FDs make sense in practice?*
Formal definition of a key

- A key is a set of attributes $A_1, ..., A_n$ s.t. for any other attribute $B$, $A_1, ..., A_n \rightarrow B$

- A minimal key is a set of attributes which is a key and for which no subset is a key

- Note: book calls them superkey and key

Examples of Keys

- Product(name, price, category, color)
  name, category $\rightarrow$ price
category $\rightarrow$ color

  Keys are: [name, category] and all supersets

- Enrollment(student, address, course, room, time)
  student $\rightarrow$ address
  room, time $\rightarrow$ course
  student, course $\rightarrow$ room, time

  Keys are: [in class]

Finding the Keys of a Relation

Given a relation constructed from an E/R diagram, what is its key?

Rules:
1. If the relation comes from an entity set, the key of the relation is the set of attributes which is the key of the entity set.

Finding the Keys

Rules:
2. If the relation comes from a many-many relationship, the key of the relation is the set of all attribute keys in the relations corresponding to the entity sets

Finding the Keys

More rules:
- Many-one, one-many, one-one relationships
- Multi-way relationships
- Weak entity sets

(Try to find them yourself, or check book)