Lecture 2:
Database Modeling (end)
The Relational Data Model
April 11, 2002

Today’s Attractions
• End of database modeling:
  – Subclasses (2.4)
  – Constraints: (2.5)
  – Weak entity sets (2.6)
• The relational data model
• From E/R diagrams to relations
• Start functional dependencies

Administration
• Exam date: June 10th, anyone?
• Project groups

Modeling Subclasses
Some objects in a class may be special
• define a new class
• better: define a subclass

Subclasses in ODL

```java
interface SoftwareProduct: Product{
    attribute string platform;
    attribute integer requiredMemory;
}
interface EducationalProduct: Product{
    attribute struct Interval {integer begin, integer end} ageGroup;
    attribute string topic;
}
```

The two classes inherit all the properties of Product.

Understanding Subclasses
• Think in terms of records:
  – Product
    - field1
    - field2
  – SoftwareProduct
    - field1
    - field2
  – EducationalProduct
    - field1
    - field2
    - field3
    - field4
    - field5

So --- we define subclasses (in ODL and in E/R).
Multiple Inheritance in ODL

interface EducSoftwareProduct:
    SoftwareProduct, EducationalProduct {
        attribute string educational-method;
    }

Understanding Multiple Inheritance

- Think in terms of records:
  - EducSoftwareProduct

How do we resolve conflicts?

Subclasses in E/R Diagrams

Difference between ODL and E/R inheritance

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

- E/R: entity sets overlap

![Diagram of overlapping entity sets]

- No need for multiple inheritance in E/R

![Diagram showing three entity sets and four kinds of objects]

Constraints

- A constraint = an assertion about the database that must be true at all times
- part of the db schema
- types in programming languages do not have anything similar
- correspond to invariants in programming languages

Modeling Constraints

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.
- **Domain constraints**: peoples’ ages are between 0 and 150.
- **General constraints**: all others (at most 50 students enroll in a class)

Keys

- **Multi-attribute keys**:
  - E.g. name + address

- **Multiple keys**:
  - E.g social-security-number, name + address
Keys in ODL

interface Person {
    (key ssn)
    attribute string ssn;
    attribute string name;
    ...}

Defining multiple keys:
(key ssn employeID (name address age))

Keys in E/R Diagrams

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

Single Value Constraints

- Sometimes we can choose to allow one or more values
- ODL:
  - attributes are always single value
  - relationships have single or multiple values
    relationship person president;
    relationship set<person> presidents;

Single Value Constraints

- E/R:

Referential Integrity Constraints

- In some formalisms we may refer to other object but get garbage instead
  - e.g. a dangling pointer in C/C++
- the Referential Integrity Constraint explicitly requires a reference to exist.
Referential Integrity Constraints

- In E/R:

  ![Diagram of Referential Integrity Constraints]

Weak Entity Sets

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

This happens if:
- part-of hierarchies
- splitting n-ary relations to binary.

The Relational Model: Outline

- The relational model (3.1)
- E/R to relational model (3.2)
- Subclasses to relational model (3.3)
- ODL to relational model (read on your own, section 4.4).

The Relational Data Model

- Database Model (ODL, E/R)
- Relational Schema
- Physical storage

<table>
<thead>
<tr>
<th>Database Model (ODL, E/R)</th>
<th>Relational Schema</th>
<th>Physical storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODL definitions</td>
<td>Tables:</td>
<td>Complex file organization and index structures.</td>
</tr>
<tr>
<td>Diagrams (E/R)</td>
<td>column names:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>attributes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rows: tuples</td>
<td></td>
</tr>
</tbody>
</table>

Terminology

<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>

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
- Relational schema = R(A1, …, Ak):
  Instance = relation with k attributes (of “type” R)
  - values of corresponding domains

Database schema = R1(…), R2(…), …, Rn(…)
Instance = n relations, of types R1, R2, ..., Rn

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

<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>

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)
- Relation = 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
  \[ A = \{ \text{name} , \text{price}, \text{category}, \text{manufacturer} \} \]
- A tuple = function \( t : A \rightarrow \text{Domains} \)
- Relation = set of tuples
- E.g.
  \[
  \begin{array}{c|c|c|c|}
  \text{name} & \text{gizmo} & \text{price} & 19 \\
  \text{category} & \text{gadgets} & \text{manufacturer} & \text{gizmoWorks} \\
  \end{array}
  \]
  - Order in a tuple is not important
  - Attribute names are important

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

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 \rightarrow \text{relations}
Relationships \rightarrow \text{relations}
Special care for weak entity sets.

Entity Sets to Relations
Product: \hspace{1cm} \text{name} \hspace{1cm} \text{category} \hspace{1cm} \text{price}

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>gizmo</td>
<td>gadgets</td>
<td>$19.99</td>
</tr>
</tbody>
</table>

Relationships to Relations
Relation Makes (watch out for attribute name conflicts)

<table>
<thead>
<tr>
<th>Product-name</th>
<th>Product-Category</th>
<th>Company-name</th>
<th>Starting-year</th>
</tr>
</thead>
<tbody>
<tr>
<td>gizmo</td>
<td>gadgets</td>
<td>gizmoWorks</td>
<td>1963</td>
</tr>
</tbody>
</table>
Many-one Relationships

No need for **Makes**. Just modify **Product**:

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Price</th>
<th>Start Year</th>
<th>Company Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>gizmo</td>
<td>gadgets</td>
<td>19.99</td>
<td>1963</td>
<td>gizmoWorks</td>
</tr>
</tbody>
</table>

Handling Weak Entity Sets

Relation: **Team**:

<table>
<thead>
<tr>
<th>Sport</th>
<th>Number</th>
<th>Affiliated University</th>
</tr>
</thead>
<tbody>
<tr>
<td>mud wrestling</td>
<td>15</td>
<td>Montezuma State U.</td>
</tr>
</tbody>
</table>

- 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 table

- **Product**: (name, price, category, manufacturer)
- **EducationalProduct**: (name, ageGroup, topic)
- **SoftwareProduct**: (name, platforms, requiredMemory)
- **EducationalSoftwareProduct**: (name, educational-method, 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.

Too many meanings for NULL
Relational Schema Design

- Normalization: general idea
- Functional dependencies: the tool
- Using FD’s for normalization.

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

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  → Name, Phone, Position
- Position → Phone
  - but Phone → Position

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:

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]

Main (and simplest) example: keys

Examples

- 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
\[ A_1, A_2, ..., A_n \]
then they must also agree on the attributes
\[ B_1, B_2, ..., B_m \]
Formally:
\[ A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m \]

Main (and simplest) example: keys
How many different FDs are there?

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 \n\rightarrow Position

In General

- To check \( A \rightarrow B \), erase all other columns
- check if the remaining relation is many-one
  (called functional in mathematics)

More Examples

Product:  name \rightarrow price, manufacturer
Person: ssn \rightarrow name, age
Company: name \rightarrow stock price, president

Key of a relation is a set of attributes that:
- functionally determines all the attributes of the relation
- none of its subsets determines all the attributes.

Superkey: a set of attributes that contains a key.

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.

          Person(address, name, ssn)

          Person

          address

          name

          Ssn
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.

But: if there is an arrow from the relationship to E, then we don’t need the key of E as part of the relation key.

Purchase(name, sname, ssn, card-no)

Finding the Keys

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

(Try to find them yourself, check book)

Finding the Keys

Rules for FD’s

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]

Splitting rule and Combing rule

Is equivalent to

\[ A_1, A_2, \ldots, A_n \rightarrow B_1 \]
\[ A_1, A_2, \ldots, A_n \rightarrow B_2 \]
\[ \vdots \]
\[ A_1, A_2, \ldots, A_n \rightarrow B_m \]

Rules in FD’s (continued)

Trivial Rule

\[ A_1, A_2, \ldots, A_n \rightarrow A_i \]

Why?

Rules in FD’s (continued)

Transitive Closure Rule

If
\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]
\[ B_1, B_2, \ldots, B_m \rightarrow C_1, C_2, \ldots, C_p \]

then
\[ A_1, A_2, \ldots, A_n \rightarrow C_1, C_2, \ldots, C_p \]

Why?
Closure of a set of Attributes

Given a set of attributes \( \{A_1, \ldots, A_n\} \) and a set of dependencies \( S \).
Problem: find all attributes \( B \) such that:
any relation which satisfies \( S \) also satisfies:
\( A_1, \ldots, A_n \rightarrow B \)

The closure of \( \{A_1, \ldots, A_n\} \), denoted \( \{A_1, \ldots, A_n\}^* \),
is the set of all such attributes \( B \)

Closure Algorithm

Start with \( X = \{A_1, \ldots, A_n\} \).

Repeat until \( X \) doesn’t change do:
if \( B \) is in \( S \), and
\( B_1, B_2, \ldots, B_n \) are all in \( X \), and
\( C \) is not in \( X \)
then
add \( C \) to \( X \).

Example

\[
\begin{align*}
A &\rightarrow B \\
C &\rightarrow D \\
E &\rightarrow F
\end{align*}
\]

Closure of \( \{A, B\} \): \( X = \{A, B, C\} \)

Why Is the Algorithm Correct?

- Show the following by induction:
  - For every \( B \) in \( X \): \( A_1, \ldots, A_n \rightarrow B \)
  - Initially \( X = \{A_1, \ldots, A_n\} \) — holds
  - Induction step: \( B_1, \ldots, B_m \) in \( X \)
    - Implies \( A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m \)
    - We also have \( B_1, \ldots, B_m \rightarrow C \)
    - By transitivity we have \( A_1, \ldots, A_n \rightarrow C \)
  - This shows that the algorithm is sound; need to show it is complete

Relational Schema Design
(or Logical Design)

Main idea:
- Start with some relational schema
- Find out its FD’s
- Use them to design a better relational schema

Relational Schema Design

Conceptual Model: [Diagram]

Relational Model: [Diagram]

Normalization: [Diagram]
Relational Schema Design
Recall set attributes (persons with several phones):

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Phone Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-321-99</td>
<td>(201) 555-1234</td>
</tr>
<tr>
<td>Fred</td>
<td>123-321-99</td>
<td>(206) 572-4312</td>
</tr>
<tr>
<td>Joe</td>
<td>909-438-44</td>
<td>(908) 464-0028</td>
</tr>
<tr>
<td>Joe</td>
<td>909-438-44</td>
<td>(212) 555-4000</td>
</tr>
</tbody>
</table>

Note: SSN no longer a key here

Goal: try to reduce anomalies:
Redundancy = repeat data
update anomalies = need to update in many places
deletion anomalies = need to delete many tuples

Relation Decomposition
Break the relation into two:

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-321-99</td>
<td>Fred</td>
</tr>
<tr>
<td>909-438-44</td>
<td>Joe</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>Phone Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-321-99</td>
<td>(201) 555-1234</td>
</tr>
<tr>
<td>123-321-99</td>
<td>(206) 572-4312</td>
</tr>
<tr>
<td>909-438-44</td>
<td>(908) 464-0028</td>
</tr>
<tr>
<td>909-438-44</td>
<td>(212) 555-4000</td>
</tr>
</tbody>
</table>

Decompositions in General
Let $R$ be a relation with attributes $A_1, A_2, \ldots, A_n$
Create two relations $R_1$ and $R_2$ with attributes

$B_1, B_2, \ldots, B_m \quad C_1, C_2, \ldots, C_1$

Such that:

$B_1, B_2, \ldots, B_m \cup C_1, C_2, \ldots, C_1 = A_1, A_2, \ldots, A_n$

And

$\longrightarrow R_1$ is the projection of $R$ on $B_1, B_2, \ldots, B_m$

$\longrightarrow R_2$ is the projection of $R$ on $C_1, C_2, \ldots, C_1$

Incorrect Decomposition

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gismo</td>
<td>Camera</td>
<td>29.99</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
<td>24.99</td>
</tr>
<tr>
<td>DoubleClick</td>
<td>Camera</td>
<td>29.99</td>
</tr>
</tbody>
</table>

Decompose on Name, Category and Price, Category

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gismo</td>
<td>Camera</td>
<td>29.99</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
<td>24.99</td>
</tr>
<tr>
<td>DoubleClick</td>
<td>Camera</td>
<td>29.99</td>
</tr>
</tbody>
</table>

When we put it back:
Cannot recover information

Normal Forms
First Normal Form = all attributes are atomic
Second Normal Form (2NF) = old and obsolete
Third Normal Form (3NF) = this lecture
Boyce Codd Normal Form (BCNF) = this lecture
Others...

Boyce-Codd Normal Form
A simple condition for removing anomalies from relations:

A relation $R$ is in BCNF if and only if:

Whenever there is a nontrivial dependency $A_1, A_2, \ldots, A_m \rightarrow B$
for $R$, it is the case that $\{ A_1, A_2, \ldots, A_m \}$

a super-key for $R$.

In English (though a bit vague):

Whenever a set of attributes of $R$ is determining another attribute,
should determine all the attributes of $R$. 
Example

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Phone Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-321-99</td>
<td>(201) 555-1234</td>
</tr>
<tr>
<td>Fred</td>
<td>123-321-99</td>
<td>(206) 572-4312</td>
</tr>
<tr>
<td>Joe</td>
<td>909-438-44</td>
<td>(908) 464-0028</td>
</tr>
<tr>
<td>Joe</td>
<td>909-438-44</td>
<td>(212) 555-4000</td>
</tr>
</tbody>
</table>

What are the dependencies?
SSN → Name
What are the keys?
Is it in BCNF?

Decompose it into BCNF

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-321-99</td>
<td>Fred</td>
</tr>
<tr>
<td>909-438-44</td>
<td>Joe</td>
</tr>
</tbody>
</table>

What are the dependencies?
SSN → Name
What are the keys?
Is it in BCNF?

What About This?

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>gadgets</td>
</tr>
<tr>
<td>OneClick</td>
<td>$24.99</td>
<td>camera</td>
</tr>
</tbody>
</table>

Name → Price, Category

BCNF Decomposition

Find a dependency that violates the BCNF condition:
A₁, A₂, …, Aₙ → B₁, B₂, …, Bₘ

Heuristics: choose B₁, B₂, …, Bₘ “as large as possible”

Decompose:
Find a 2-attribute relation that is not in BCNF.

Continue until there are no BCNF violations left.

Example Decomposition

Person:

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Age</th>
<th>EyeColor</th>
<th>PhoneNumber</th>
</tr>
</thead>
</table>

Functional dependencies:
SSN → Name, Age, EyeColor

BNCF:
Person₁(SSN, Name, Age, EyeColor),
Person₂(SSN, PhoneNumber)

What if we also had an attribute Draft-worthy, and the FD:
Age → Draft-worthy