CSE544
Data Modeling, Conceptual Design
Wednesday, April 7, 2004

Outline
• ER diagrams (Chapter 2)
• Conceptual Design (Chapter 19)

Database Design
Data Modeling → Refinement → SQL Tables → Files
E/R diagrams → Relations

Relational Schema Design
Conceptual Model

Relational Model: plus FD’s

Normalization: Eliminates anomalies

Entity / Relationship Diagrams

Attributes
Entity sets
Relationships

Person
Company
Product

makes

employees

name

name

name

buy

Product

buys

Person

buys

buys

buys

stock_price
Keys in E/R Diagrams

- Every entity set must have a key

Multiplicity of E/R Relations

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

Multi-way Relationships

Arrows in Multiway Relationships

Q: What does the arrow mean?

A: If I know the store, person, invoice, I know the movie too

Arrows in Multiway Relationships

Q: What do these arrows mean?

A: Store, person, invoice determines movie; and store, invoice, movie determines person
Arrows in Multiway Relationships

Q: how do I say: “invoice determines store”?  
A: no good way; best approximation:

Incomplete (why?)

Roles in Relationships

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

Attributes on Relationships

Converting Multi-way Relationships to Binary

Need arrows here!  Which direction?

From E/R Diagrams to Relational Schema

• Entity set relation
• Relationship relation

Entity Set to Relation

<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**

```
Makes(product-name, product-category, company-name, year)

(watch out for attribute name conflicts)
```

**Multi-way Relationships to Relations**

```
Product

<table>
<thead>
<tr>
<th>name</th>
<th>price</th>
<th>address</th>
</tr>
</thead>
</table>

Purchase(prodName, ssn)

Store

Person

<table>
<thead>
<tr>
<th>name</th>
<th>ssn</th>
</tr>
</thead>
</table>
```

**Design Principles**

**What’s Wrong?**

```
Moral: pick the right kind of entities.
```

**Design Principles**

**What’s Wrong?**

```
Moral: don’t complicate life more than it already is.
```
Subclasses

Product

name

category

price

Software Product

is a

Educational Product

isa

platforms

isa

Age Group

Subclasses to Relations

Product

name

category

price

Software Product

isa

Gizmo

platforms

unix

Camara

photo

Toy

gadget

Ed.Product

Name

Age Group

Gizmo
					
todler

Toy
					
retired

Notice: subclass = subset
Alternative: disjoint classes (Java, C++)

Modeling Union Types With Subclasses

FurniturePiece

Person

Company

Each piece of furniture is owned either by a person, or by a company

Modeling Union Types with Subclasses

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

Person

FurniturePiece

Company

ownedByPerson

ownedByPerson

Modeling Union Types with Subclasses

Solution 2: better

Owner

isa

Person

ownedBy

Company

FurniturePiece

Referential Integrity Constraints

Product

makes

Company

Each product made by at most one company. Some products made by no company

Product

makes

Company

Each product made by exactly one company.
Other Constraints

- Product <100 makes Company

What does this mean?

Weak Entity Sets

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

- Team affiliation University
  - sport
  - number
  - name

University(name)
Team(universityName, number, sport)

Weak Entity Sets

E1 E3

A3 A1

B3 R1

R2 E2

What are the keys?

Schema Refinement

- For the relational model
- Relation: R(A1, A2, ..., An)
  - Schema: relation name, attribute names
  - Instance: a mathematical m-ary relation
- Database: R1, R2, ..., Rn
  - Schema
  - Instance
- Schema refinement = normalization

First Normal Form (1NF)

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

<table>
<thead>
<tr>
<th>Student</th>
<th>GPA</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>3.8</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>3.7</td>
<td></td>
</tr>
<tr>
<td>Carol</td>
<td>3.9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Course</th>
<th>Math</th>
<th>Stats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Carol</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

May need to add keys

More Normal Forms

- Based on Functional Dependencies
  - 2nd Normal Form (obsolete)
  - 3rd Normal Form
  - Boyce Codd Normal Form (BCNF)
- Based on Multivalued Dependencies
  - 4th Normal Form
- Based on Join Dependencies
  - 5th Normal Form
Functional Dependencies

• A form of constraint
  – hence, part of the schema
• Finding them is part of the database design

Functional Dependencies

Meaning:

If two tuples agree on the attributes

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

then they must also agree on the attributes

\[ B_1, B_2, \ldots, B_m \]

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  
  • but Phone \( \not\) Position

Example

Product(name, category, color, department, price)

Consider these FDs:

<table>
<thead>
<tr>
<th>name</th>
<th>color</th>
<th>category</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Tweaker</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>99</td>
</tr>
</tbody>
</table>

What do they say ?

Example

name | color | category | department | price |
-----|-------|----------|------------|-------|
Gizmo| Gadget| Green    | Toys       | 49    |
Tweaker| Gadget | Green    | Toys       | 99    |

Does this instance satisfy all the FDs ?
**Example**

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Tweeker</td>
<td>Gadget</td>
<td>Black</td>
<td>Toys</td>
<td>99</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Stationary</td>
<td>Green</td>
<td>Office-sup.</td>
<td>59</td>
</tr>
</tbody>
</table>

What about this one?

**Inference**

If some FDs are satisfied, then others are satisfied too:

If all these FDs are true:

Then this FD also holds:

Why?? We say that the new FD is implied.

**Armstrong’s Axioms**

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

Is equivalent to

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

Splitting rule

And Combing rule

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

\[ B_1, B_2, \ldots, B_m \]

**Armstrong’s Axioms**

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

Trivial Rule

\[ \text{where } i = 1, 2, \ldots, n \]

Why??

**Armstrong’s Axioms**

Transitive Closure Rule

If

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

and

\[ B_1, B_2, \ldots, B_m, C_1, C_2, \ldots, C_p \]

then

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

Why??

**Example (continued)**

From:

1. name, color
2. category, department
3. color, category

To:

name, category, price

Inferred FD

Which Rule did we apply?

<table>
<thead>
<tr>
<th>Inferred FD</th>
<th>Which Rule did we apply?</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. name, category, name</td>
<td></td>
</tr>
<tr>
<td>5. name, category, color</td>
<td></td>
</tr>
<tr>
<td>6. name, category, category</td>
<td></td>
</tr>
<tr>
<td>7. name, category, color, category</td>
<td></td>
</tr>
<tr>
<td>8. name, category, price</td>
<td></td>
</tr>
</tbody>
</table>
Example (continued)

Answers:

<table>
<thead>
<tr>
<th>Inferred FD</th>
<th>Which Rule did we apply?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. name color</td>
<td>Trivial rule</td>
</tr>
<tr>
<td>2. category department</td>
<td>Transitivity on 3, 7</td>
</tr>
<tr>
<td>3. color, category price</td>
<td>Transitivity on 4, 1</td>
</tr>
</tbody>
</table>

| 4. name, category name | Trivial rule |
| 5. name, category color | Transitivity on 4, 1 |
| 6. name, category category | Transitivity on 4, 1 |
| 7. name, category color, category | Split/combine on 5, 6 |
| 8. name, category price | Transitivity on 3, 7 |

Closure of a Set of FDs

**Definition.** Given a set $F$ of functional dependencies, the closure, $F^+$, denotes all FDs implied by $F$.

**Theorem.** Armstrong axioms are *sound* and *complete* for computing $F^+$.

What do *sound* and *complete* mean?

Variation

**Augmentation**

If

$A_1, A_2, \ldots, A_n \rightarrow B$

then

$A_1, A_2, \ldots, A_n, C_1, C_2, \ldots, C_p \rightarrow B$

Augmentation follows from trivial rules and transitivity

How?

Problem: Compute $F^+$

Given $F$ compute its closure $F^+$.

How to proceed?

- Apply Armstrong’s Axioms repeatedly
- Better: use the Closure Algorithm for a set of attributes (next)

Closure of a Set of Attributes

Given a set of attributes $A_1, \ldots, A_n$.

The closure, $\{A_1, \ldots, A_n\}^+$, is the set of attributes $B$ s.t. $A_1, \ldots, A_n \rightarrow B$.

Example:

<table>
<thead>
<tr>
<th>name color category department color, category price</th>
</tr>
</thead>
<tbody>
<tr>
<td>[name, color]</td>
</tr>
</tbody>
</table>

Closures:

- $\text{name}^+ = \{\text{name}, \text{color}\}$
- $\{\text{name}, \text{category}\}^+ = \{\text{name}, \text{category}, \text{color}, \text{department}, \text{price}\}$
- $\text{color}^+ = \{\text{color}\}$

Closure Algorithm (for Attributes)

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

Repeat until $X$ doesn’t change:

if $B_1, \ldots, B_n \rightarrow C$ is a FD and $B_1, \ldots, B_n \in X$ then add $C$ to $X$.

Example:

name color category department color, category price

$\{\text{name}, \text{category}\}^+ = \{\text{name}, \text{category}, \text{color}, \text{department}, \text{price}\}$
Example

In class:

\[ R(A,B,C,D,E,F) \]

\[
\begin{array}{cccc}
A & B & C \\
A & D & E \\
B & D & \\
A & F & B \\
\end{array}
\]

Compute \( (A,B)^+ \quad X = \{ A, B, \} \)

Compute \( (A,F)^+ \quad X = \{ A, F, \} \)

Closure Algorithm (for FDs)

Example:

\[
\begin{array}{cccc}
A & B & C \\
A & D & B \\
B & D & \\
\end{array}
\]

Step 1: Compute \( X^+ \) for every \( X \):

\[ A^+ = A, \quad B^+ = BD, \quad C^+ = C, \quad D^+ = D \]
\[ AB^+ = ABCD, \quad AC^+ = AC, \quad AD^+ = ABCD \]
\[ ABC^+ = ABD^+ = ACD^+ = ABCD \text{ (no need to compute-- why?)} \]
\[ BCD^+ = BCD, \quad ABCD^+ = ABCD \]

Step 2: Enumerate all FD’s \( X \quad Y \), s.t. \( Y \subseteq X \) and \( X \cap Y = \emptyset \):

\[ AB \quad CD, \quad AD \quad BC, \quad ABC \quad D, \quad ABD \quad C, \quad ACD \quad B \]

Keys

- A **superkey** is a set of attributes \( A_1, ..., A_n \) s.t. \( A_1, ..., A_n \Rightarrow B \) for all attributes \( B \)

- A **key** is a minimal superkey

Computing Keys

- Compute \( X^+ \) for all sets \( X \)
- If \( X^+ = \) all attributes, then \( X \) is a superkey
- Consider only the minimal superkeys

Note: there can be exponentially many keys!
- Example: \( R(A,B,C) \), \( AB \quad C, \quad BC \) \( A \)

Keys: \( AB \) and \( BC \)

Examples of Keys

- \( \text{Product} \text{ (name, price, category, color) } \)
  - name, category \quad price
  - category \quad color

  Key: \{ name, category \} \quad Superkeys: supersets

- \( \text{Enrollment} \text{ (student, address, course, room, time) } \)
  - student, address
  - course, room, time

  Keys are: \{ in class \}

FD’s for E/R Diagrams

Say: “the CreditCard determines the Person”

\[ \text{Incomplete (what does it say ?)} \]

\[ \text{Purchase} \text{ (name, sname, ssn, card-no) } \]

\[ \text{card-no, ssn} \]
Data Anomalies

When a database is poorly designed we get anomalies:

- **Redundancy**: data is repeated
- **Updated anomalies**: need to change in several places
- **Delete anomalies**: may lose data when we don’t want

Schema refinement means removing the data anomalies.

---

Relation Decomposition

Break the relation into two:

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

Anomalies have gone:

- No more repeated data
- Easy to move Fred to “Bellevue” (how?)
- Easy to delete all Joe’s phone number (how?)

---

Decompositions in General

\[ R(A_1, ..., A_n, B_1, ..., B_m, C_1, ..., C_p) \]

\[ R_1(A_1, ..., A_n, B_1, ..., B_m) \]

\[ R_2(A_1, ..., A_n, C_1, ..., C_p) \]

\[ R_1 = \text{projection of } R \text{ on } A_1, ..., A_n, B_1, ..., B_m \]

\[ R_2 = \text{projection of } R \text{ on } A_1, ..., A_n, C_1, ..., C_p \]

---

Problems With Decomposition

- Can we get the data back correctly?
  - Lossless decomposition
  - Discuss next
- Can we recover the FD’s on the ‘big’ table from the FD’s on the small tables?
  - Dependency-preserving decomposition
  - Figure out yourself, or read 19.5.2

---

Lossless Decomposition

- Sometimes it is correct:

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gismo</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gismo</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
</tbody>
</table>
Lossy Decomposition

- Sometimes it is not:

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClickCamera</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

What’s wrong??

Decompositions in General

\[
R(A_1, ..., A_n, B_1, ..., B_m, C_1, ..., C_p)
\]

\[
R_1(A_1, ..., A_n, B_1, ..., B_m)
\]

\[
R_2(A_1, ..., A_n, C_1, ..., C_p)
\]

- If \( A_1, ..., A_n \) \( \rightarrow \) \( B_1, ..., B_m \) is a non-trivial dependency in \( R \), then \( \{ A_1, ..., A_n \} \) is a superkey for \( R \)

Theorem: If \( A_1, ..., A_n \) \( \rightarrow \) \( B_1, ..., B_m \)

Then the decomposition is lossless

Note: don’t need necessarily \( A_1, ..., A_n \) \( \rightarrow \) \( C_1, ..., C_p \)

- Example: name price, hence the first decomposition is lossless

Normal Forms

- Decomposition into Boyce Codd Normal Form (BCNF)
  - Lossless

- Decomposition into 3rd Normal Form
  - Lossless
  - Dependency preserving

Boyce-Codd Normal Form

A simple condition for removing anomalies from relations:

A relation \( R \) is in BCNF if:

- If \( A_1, ..., A_n \) \( \rightarrow \) \( B_1, ..., B_m \) is a non-trivial dependency in \( R \), then \( \{ A_1, ..., A_n \} \) is a superkey for \( R \)

Equivalently: for any set of attributes \( X \), either \( X^+ = X \)

or \( X^+ = \) all attributes

BCNF Decomposition Algorithm

Repeat

- choose \( A_1, ..., A_n \) \( \rightarrow \) \( B_1, ..., B_m \) that violates the BCNF condition
- split \( R \) into \( R_1(A_1, ..., A_n, B_1, ..., B_m) \) and \( R_2(A_1, ..., A_n, \) rest\)

Until no more violations

Heuristics: choose \( B_1, ..., B_m \) “as large as possible”

Note: need to compute the FDs on \( R_1, R_2 \) (how?)

Is there a 2-attribute relation that is not in BCNF?

BCNF Example

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>197-55-6321</td>
<td>908-555-2123</td>
<td>Westfield</td>
</tr>
<tr>
<td>Joe</td>
<td>197-55-6321</td>
<td>908-555-1234</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

FD: SSN, Name, City

Key: \{ SSN, PhoneNumber \}

Is it in BCNF?

Another way: \( SSN^+ = \{ SSN, Name, City \} \) but no PhoneNumber
**Example**

- R(A,B,C,D) \(\rightarrow\) A, B, B, C
- Key: AD
- Violations of BCNF:
  - A \(\rightarrow\) B, C
  - A \(\rightarrow\) B
  - B \(\rightarrow\) C

- Pick A BC first: split into \(R_1(A,B,C)\) \(\rightarrow\) (A,D)
- In \(R_1\) : B \(\rightarrow\) C, split into \(R_{11}(A,B)\) \(\rightarrow\) (B,C)
- Final answer: \(R_{11}(A,B), R_{12}(B,C), R_2(A,D)\)

**Solution: 3rd Normal Form (3NF)**

A simple condition for removing anomalies from relations:

\[ A \text{ relation R is in 3rd normal form if:} \]

Whenever there is a nontrivial dependency \(A_1, A_2, ..., A_n \rightarrow B\)
for \(R\), then \(\{A_1, A_2, ..., A_n\}\) a super-key for \(R\),
or \(B\) is part of a key.

Please read in the book !!!