The Relational Data Model

Lecture 6

Outline

• Relational Data Model
• Functional Dependencies
• Logical Schema Design

Reading
Chapter 8
The Relational Data Model

Data Modeling → Relational Schema → Physical storage

E/R diagrams → Tables: column names: attributes rows: tuples

Complex file organization and index structures.

Have seen this in SQL

Have seen this too

Discuss next

Terminology

Table name or relation name

Products:

<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>

Tuples 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!

(What's the difference?)

Instances

- **Relational schema** = \( R(A_1, \ldots, A_k) \)
  
  **Instance** = relation with \( k \) attributes

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

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

(What's the difference?)
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>

Design Criteria

- A relational schema should ensure:
  - Data integrity
    - data is consistent and satisfies integrity constraints
  - Data redundancy should be avoided
- The process of creating a relational schema that follows certain rules is called normalisation
  - We will learn several of these rules
Another “Example”

Student

<table>
<thead>
<tr>
<th>Name</th>
<th>GPA</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>3.8</td>
<td>Math</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OS</td>
</tr>
<tr>
<td>Bob</td>
<td>3.7</td>
<td>DB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OS</td>
</tr>
<tr>
<td>Carol</td>
<td>3.9</td>
<td>Math</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OS</td>
</tr>
</tbody>
</table>

How can this be expressed in a relational schema?

Outlook: First Normal Form (1NF)

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

Student

<table>
<thead>
<tr>
<th>Name</th>
<th>GPA</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>3.8</td>
<td>Math</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OS</td>
</tr>
<tr>
<td>Bob</td>
<td>3.7</td>
<td>DB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OS</td>
</tr>
<tr>
<td>Carol</td>
<td>3.9</td>
<td>Math</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OS</td>
</tr>
</tbody>
</table>

Takes

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Math</td>
</tr>
<tr>
<td>Carol</td>
<td>Math</td>
</tr>
<tr>
<td>Alice</td>
<td>DB</td>
</tr>
<tr>
<td>Bob</td>
<td>DB</td>
</tr>
<tr>
<td>Alice</td>
<td>OS</td>
</tr>
<tr>
<td>Carol</td>
<td>OS</td>
</tr>
</tbody>
</table>

Course

<table>
<thead>
<tr>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
</tr>
<tr>
<td>DB</td>
</tr>
<tr>
<td>OS</td>
</tr>
</tbody>
</table>
Theoretically, also this flattened table is in 1NF but it has several problems:

- Update anomalies
- Inserts etc.

Only a compound key between Name and Course is possible.

### 1NF cont.

**Features:**

- All values (attributes) are atomic
  - For instance, no comma separated are values allowed!
- “Conventional” SQL based databases typically adhere to the 1NF

**Relational Rule 1** (based on Codd’s 12 Rules):

- *A table that has no multi-valued fields is said to be in the first normal form.*
Normal Forms: Overview

- 1st Normal Form (1NF)
- 2nd Normal Form (2NF)
- 3rd Normal Form (3NF)
- Boyce Codd Normal Form (BCNF)

• The higher the normal form, the more redundancies are reduced

Outline

• Relational Data Model
• Functional Dependencies
• Logical Schema Design
Functional Dependencies (FD)

- A form of constraint
  - hence, part of the schema
- Finding them is part of the database design
- Also used in normalising the relations

*Warning: this is the most abstract, and “hardest” part of the course.*

Functional Dependency: Graphically

(Patrick O’Neil 1994)
FD cont.

• Important: the intent of the DB designer is expressed
• “Two rows cannot agree in value on attribute A and disagree on B”.

\[
\text{If } r_1(A) = r_2(A) \text{ then } r_1(B) = r_2(B)
\]

– “A functionally determines B”
– “B is functionally dependent on A”

• In other words: A must be unique

FD Definitions

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 $\nrightarrow$ Position

Example

<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>

Position $\rightarrow$ Phone
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-to-one (called functional in mathematics)

Typical Examples of FDs

Product: name $\rightarrow$ price, manufacturer

Person: ssn $\rightarrow$ name, age

Company: name $\rightarrow$ stockprice, president
Example

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

Consider these FDs:

- name → color
- category → department
- color, category → price

What do they say?

Example

FDs are constraints on relations:
- On some instances they hold
- On others they don’t

<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>Tweaker</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>99</td>
</tr>
</tbody>
</table>

Does this instance satisfy all the FDs?
Example

If some FDs are satisfied, then others are satisfied too

If all these FDs are true:

Then this FD also holds:

Why ??
Inference Rules for FDs

(1) Splitting rule  
and  
(2) Combining rule

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

Is equivalent to

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

Inference Rules for FDs (continued)

(3) Trivial Rule

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

where \(i = 1, 2, \ldots, n\)

\(A_i\) is a subset of \(A_{1..n}\)
Inference Rules for FDs (continued)

(4) Transitive Closure Rule

If \[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]
and \[ B_1, B_2, \ldots, B_m \rightarrow C_1, C_2, \ldots, C_p \]
then \[ A_1, A_2, \ldots, A_n \rightarrow C_1, C_2, \ldots, C_p \]
Example (continued)

Start from the following FDs:

1. name $\rightarrow$ color
2. category $\rightarrow$ department
3. color, category $\rightarrow$ price

Infer the following FDs:

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

Example (continued)

Answers:

1. name $\rightarrow$ color
2. category $\rightarrow$ department
3. color, category $\rightarrow$ price

<table>
<thead>
<tr>
<th>Inferred FD</th>
<th>Which Rule did we apply ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. name, category $\rightarrow$ name</td>
<td>Trivial rule</td>
</tr>
<tr>
<td>5. name, category $\rightarrow$ color</td>
<td>Transitivity on 4, 1</td>
</tr>
<tr>
<td>6. name, category $\rightarrow$ category</td>
<td>Trivial rule</td>
</tr>
<tr>
<td>7. name, category $\rightarrow$ color, category</td>
<td>Split/combine on 5, 6</td>
</tr>
<tr>
<td>8. name, category $\rightarrow$ price</td>
<td>Transitivity on 3, 7</td>
</tr>
</tbody>
</table>
Another Example

- Enrollment(student, major, course, room, time)
  - student → major
  - major, course → room
  - course → time

Another Rule

(5) Augmentation Rule

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?
“Solving” the Augmentation Rule

Trivial Rule

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

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

Transitivity gives:

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

Summary of Rules

(1) Splitting (Decomposition) Rule

If \( X \rightarrow YZ \), then \( X \rightarrow Y \) and \( X \rightarrow Z \)

(2) Combining (Union) Rule

If \( X \rightarrow Y \) and \( X \rightarrow Z \), then \( X \rightarrow YZ \)

(3) Trivial (Reflexivity) Rule

If \( Y \subseteq X \), then \( X \rightarrow Y \)

(4) Transitive Closure Rule

If \( X \rightarrow Y \) and \( Y \rightarrow Z \), then \( X \rightarrow Z \)

(5) Augmentation Rule

If \( X \rightarrow Y \), then \( XZ \rightarrow Y \), for every \( Z \)
Problem: infer ALL FDs

Given a set of FDs, infer all possible FDs

How to proceed?
• Try all possible FDs, apply all rules
  – E.g. R(A, B, C, D): how many FDs are possible?
    • Answer: $2^4$ subsets of attributes
• Drop trivial FDs, drop augmented FDs
  – Still way too many
• Better: use the Closure Algorithm (next)

FDs and Closure

• Typically, a relation R has a set of defined functional dependencies F
• The closure of F (written $F^+$) is the set of all functional dependencies that may be derived from F
  – $F^+$ contains all defined FDs and derived FDs
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:

\[
\begin{align*}
\text{name} & \rightarrow \text{color} \\
\text{category} & \rightarrow \text{department} \\
\text{color, category} & \rightarrow \text{price}
\end{align*}
\]

Closures:

\[
\begin{align*}
\text{name}^+ & = \{\text{name, color}\} \\
\{\text{name, category}\} & \rightarrow \{\text{name, category, color, department, price}\} \\
\text{color}^+ & = \{\text{color}\}
\end{align*}
\]

Closure Algorithm

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

**Repeat until** \( X \) doesn’t change do:

- if \( B_1, \ldots, B_n \rightarrow C \) is a FD and \( B_1, \ldots, B_n \) are all in \( X \) then add \( C \) to \( X \).

Example:

\[
\begin{align*}
\text{name} & \rightarrow \text{color} \\
\text{category} & \rightarrow \text{department} \\
\text{color, category} & \rightarrow \text{price}
\end{align*}
\]

\[
\{\text{name, category}\}^+ = \{\text{name, category, color, department, price}\}
\]
Closure Algorithm more verbose

- Starting with the given set of attributes, repeatedly expand the set by adding the right sides of FDs as soon as we have included their left sides.
- Eventually, we cannot expand the set any more, and the resulting set is the closure.

1. Let $X$ be a set of attributes that eventually will become the closure. First we initialize $X$ to be $\{A_1, A_2, \ldots, A_n\}$.
2. Now, repeatedly search for some FD in $X$: $B_1, B_2, \ldots, B_m \rightarrow C$
   such that all of $B$s are in the set $X$, but $C$ is not. We then add $C$ to $X$.
3. Repeat step 2 as many times as necessary until no more attributes can be added to $X$.
   Since $X$ can only grow, and the number of attributes is finite, eventually nothing more can be added to $X$.
4. The set $X$ after no more attributes can be added to it is the: $\{A_1, A_2, \ldots, A_n\}^+. \hspace{1cm} (Alex Thono 2006)$

Example

$R(A,B,C,D,E,F)$

| A, B → C |
| A, D → E |
| B → D |
| A, F → B |

Compute $\{A,B\}^+$ \hspace{0.5cm} $X = \{A, B, \quad ? \}$

Compute $\{A, F\}^+$ \hspace{0.5cm} $X = \{A, F, \quad ? \}$
Example (Solution)

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

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

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

Compute \( \{A, F\}^+ \quad X = \{A, F, B, C, D, E\} \)

Using Closure to Infer ALL FDs

Example:

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

\[
\begin{align*}
A^+ & = A, \quad B^+ = BD, \quad C^+ = C, \quad D^+ = D \\
AB^+ & = ABCD, \quad AC^+ = AC, \quad AD^+ = ABCD \\
ABC^+ & = ABD^+ = ACD^+ = ABCD \text{ (no need to compute– why ?)} \\
BCD^+ & = BCD, \quad ABCD^+ = ABCD
\end{align*}
\]
Problem: Finding FDs

• Approach 1: During Database Design
  – Designer derives them from real-world knowledge of users
  – Problem: knowledge might not be available
• Approach 2: From a Database Instance
  – Analyze a given database instance and find all FDs satisfied by that instance
  – Useful if designers don’t get enough information from users
  – Problem: FDs might be artificial for the given instance

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>
Some Answers

\[
\begin{align*}
\text{Course} & \rightarrow \text{Dept, Room} \\
\text{Dept, Room} & \rightarrow \text{Course} \\
\text{Student, Dept} & \rightarrow \text{Course, Room} \\
\text{Student, Course} & \rightarrow \text{Dept, Room} \\
\text{Student, Room} & \rightarrow \text{Dept, Course}
\end{align*}
\]

Do all FDs make sense in practice?

Keys

- A **key** is a set of attributes \( A_1, \ldots, A_n \) s.t. for any other attribute \( B \), we have \( A_1, \ldots, A_n \rightarrow B \)
  - Example \[ A_1, A_2, A_3, A_4 \]
- A **minimal key** is a set of attributes which is a key and for which no subset is a key
  - Example \[ A_1, A_2, A_3, A_4 \]

- Note: our course book calls them **superkey** and **key**
Computing Keys

- Compute \( X^+ \) for all sets \( X \)
- If \( X^+ = \) all attributes, then \( X \) is a key
- List only the minimal keys

Note: there can be many minimal keys!

- Example: \( R(A,B,C) \), \( AB \rightarrow C \), \( BC \rightarrow A \)
  Minimal keys: \( AB \) and \( BC \)

Let’s reuse the Closure Example

Example:

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

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

Minimal keys

Keys
More Examples of Keys

• **Product(name, price, category, color)**
  
  name, category $\rightarrow$ price  
  category $\rightarrow$ color

  Keys are: $\{\text{name, category}\}$ and all supersets

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

  … done in class

Outline

• Relational Data Model
• Functional Dependencies
• Logical Schema Design
Relational Schema Design (or Logical Schema Design)

Main idea:
  • Start with some relational schema
  • Find out its FDs
  • Use them to design a better relational schema

Data Anomalies

When a database is poorly designed we get anomalies:

**Redundancy**: data is repeated

**Update anomalies**: need to change in several places

**Delete anomalies**: may lose data when we don’t want
Relational Schema Design

Example: Persons with several phones

<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>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

\[ \text{SSN} \rightarrow \text{Name, City} \quad \text{but not} \quad \text{SSN} \rightarrow \text{PhoneNumber} \]

Anomalies:
- Redundancy = repeat data
- Update anomalies = Fred moves to “Bellevue”
- Deletion anomalies = Joe deletes his phone number: what is his city?

Relation Decomposition

Break the relation into two:

<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>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

<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>

\[ \text{SSN} \rightarrow \text{PhoneNumber} \]

Anomalies have gone:
- No more repeated data
- Easy to move Fred to “Bellevue” (how?)
- Easy to delete all Joe’s phone number (how?)
Relational Schema Design

Conceptual Model:

Relational Model: plus FDs

Normalization: Eliminates anomalies

Decompositions in General

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

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

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

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

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

• Sometimes it is correct:

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

Lossless decomposition

Incorrect 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>Camera</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

Lossy decomposition

What’s incorrect ??
Decompositions in General

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 $\rightarrow$ price, hence the first decomposition is lossless

Reminder: Normal Forms

First Normal Form = all attributes are atomic

Second Normal Form (2NF) = in principle old/obsolete

Third Normal Form (3NF) = this lecture

Boyce Codd Normal Form (BCNF) = this lecture

Others...
Third Normal Form (3NF)

• A relation is in 3NF if, and only if:
  – 1NF is satisfied
  – Every non-key attribute is functionally dependent on the whole key
    • i.e. no partial key functional dependencies
  – No transitive functional dependencies:
    • A non-key attribute must not be functionally dependent on another non-key attribute
  – No data redundancies

Example

<table>
<thead>
<tr>
<th>project_no</th>
<th>manager</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>Black,B</td>
<td>32 High Street</td>
</tr>
<tr>
<td>p2</td>
<td>Smith,J</td>
<td>11 New Street</td>
</tr>
<tr>
<td>p3</td>
<td>Black,B</td>
<td>32 High Street</td>
</tr>
<tr>
<td>p4</td>
<td>Black,B</td>
<td>32 High Street</td>
</tr>
</tbody>
</table>

1NF but not 3NF
manager → address

• Now we have reached 3NF

<table>
<thead>
<tr>
<th>Project</th>
<th>project_no</th>
<th>manager</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p1</td>
<td>Black,B</td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>Smith,J</td>
</tr>
<tr>
<td></td>
<td>p3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p4</td>
<td>Black,B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Manager</th>
<th>manager</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Black,B</td>
<td>32 High Street</td>
</tr>
<tr>
<td></td>
<td>Smith,J</td>
<td>11 New Street</td>
</tr>
</tbody>
</table>

(manager, address)

(http://db.grussel.org)
Potential Problems with 3NF

• If a relation has more than 1 candidate key, anomalies may occur
• 3NF does not deal satisfactory with overlapping candidate keys
• Need a stricter form:
  – Boyce Codd Normal Form (BCNF)
• Example see:
  http://www.answers.com/topic/boyce-codd-normal-form
Boyce-Codd Normal Form

A relation R is in BCNF if:

If $A_1, \ldots, A_n \rightarrow B$ is a non-trivial dependency
in R, then $\{A_1, \ldots, A_n\}$ is a key for R

In English (though a bit vague):

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

Every determinant (LHS of the FD) is a candidate key.

BCNF Decomposition Algorithm

Repeat
choose $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ that violates the BCNF condition
split R into $R_1(A_1, \ldots, A_m, B_1, \ldots, B_n)$ and $R_2(A_1, \ldots, A_m, \text{[others]})$
continue with both $R_1$ and $R_2$
Until no more violations
**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>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-1234</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

What are the dependencies?
- SSN → Name, City

What are the keys?
- \{SSN, PhoneNumber\}

Is it in BCNF?

**Decompose it into BCNF**

<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>

Let’s check anomalies:
- Redundancy ?
- Update ?
- Delete ?
**Summary of BCNF Decomposition**

Find a dependency that violates the BCNF condition:

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

Heuristics: choose \( B_1, B_2, \ldots, B_m \) “as large as possible”

Decompose:

- \( B \)'s
- \( A \)'s
- Others

Continue until there are no BCNF violations left.

2-attribute relations are BCNF

---

**Example Decomposition**

Person(name, SSN, age, hairColor, phoneNumber)

- SSN \( \rightarrow \) name, age
- age \( \rightarrow \) hairColor

Decompose in BCNF (in class):

Step 1: find all keys (How? Compute \( S^+ \), for various sets \( S \))

Step 2: now decompose
Other Example

- \( R(A,B,C,D) \)  \( A \rightarrow B, \ B \rightarrow C \)

- Key: AD
- Violations of BCNF (need to decompose)
  - \( A \rightarrow B, \ A \rightarrow C, \ A \rightarrow BC \)
- Pick \( A \rightarrow BC \):
  - split into \( R_1(A,B,C) \) \( R_2(A,D) \)
- What happens if we pick \( A \rightarrow B \) first?

Lossless Decompositions

A decomposition is \textit{lossless} if we can recover:

\[ R(A,B,C) \]

Decompose

\[ R_1(A,B) \]

\[ R_2(A,C) \]

Recover

\[ R'(A,B,C) \]

should be the same as \( R(A,B,C) \)

\( R' \) is in general larger than \( R \). Must ensure \( R' = R \)
Lossless Decompositions

• Given R(A,B,C) s.t. A → B, the decomposition into R1(A,B), R2(A,C) is lossless

3NF: A Problem with BCNF

<table>
<thead>
<tr>
<th>Unit</th>
<th>Company</th>
<th>Product</th>
</tr>
</thead>
</table>

FDs: Unit → Company; Company, Product → Unit
So, there is a BCNF violation, and we decompose.

<table>
<thead>
<tr>
<th>Unit</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unit</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notice: we loose the FD: Company, Product → Unit
So What’s the Problem?

<table>
<thead>
<tr>
<th>Unit</th>
<th>Company</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Galaga99</td>
<td>UW</td>
<td>databases</td>
</tr>
<tr>
<td>Bingo</td>
<td>UW</td>
<td>databases</td>
</tr>
</tbody>
</table>

No problem so far. All *local* FD’s are satisfied.
Let’s put all the data back into a single table again (anomalies?):

<table>
<thead>
<tr>
<th>Unit</th>
<th>Company</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Galaga99</td>
<td>UW</td>
<td>databases</td>
</tr>
<tr>
<td>Bingo</td>
<td>UW</td>
<td>databases</td>
</tr>
</tbody>
</table>

*Violates the dependency: company, product -> unit!*  

Solution: 3rd Normal Form (3NF)

A relation R is in Third Normal Form if:

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

Tradeoff:

BCNF = no anomalies, but may lose some FDs
3NF = keeps all FDs, but may have some anomalies
Summary

• Dependencies on attributes are important when designing database schema
  – Functional dependencies
  – Attributes should be dependent only on (primary) key
• Use normalised database schemas to avoid certain anomalies with data