Introduction to Data Management
CSE 344

Unit 6: Conceptual Design
E/R Diagrams
Integrity Constraints
BCNF

(3 lectures)
Introduction to Data Management
CSE 344

E/R Diagrams
Announcements

• HW6 due tonight. Turn instances off!!!

• WebQuiz 6 due on Wednesday

• HW7 posted, due next Friday
Class Overview

• Unit 1: Intro
• Unit 2: Relational Data Models and Query Languages
• Unit 3: Non-relational data
• Unit 4: RDMBS internals and query optimization
• Unit 5: Parallel query processing
• Unit 6: DBMS usability, conceptual design
  – E/R diagrams
  – Constraints
  – Schema normalization
• Unit 7: Transactions
• Unit 8: Advanced topics (time permitting)
Database Design

What it is:

• Starting from scratch, design the database schema: relation, attributes, keys, foreign keys, constraints etc

Why it’s hard

• The database will be in operation for a very long time (years). Updating the schema while in production is very expensive (why?)
Database Design

• Consider issues such as:
  – What entities to model
  – How entities are related
  – What constraints exist in the domain

• Several formalisms exists
  – We discuss E/R diagrams
  – UML, model-driven architecture

• Reading: Sec. 4.1-4.6
Database Design Process

Conceptual Model:

Relational Model: Tables + constraints And also functional dep.

Normalization: Eliminates anomalies

Conceptual Schema

Physical storage details

Physical Schema
Entity / Relationship Diagrams

- Entity set = a class
  - An entity = an object

- Attribute

- Relationship
Keys in E/R Diagrams

• Every entity set must have a key
What is a Relation?

• A mathematical definition:
  – if A, B are sets, then a relation R is a subset of A \times B

• A={1,2,3}, B={a,b,c,d},
  \[ A \times B = \{(1,a),(1,b),(1,c),(1,d), (2,a),(2,b),(2,c),(2,d), (3,a),(3,b),(3,c),(3,d)\} \]
  \[ R = \{(1,a), (1,c), (3,b)\} \]

• \textbf{makes} is a subset of \textbf{Product} \times \textbf{Company}:

![Diagram showing a relation between Product and Company with elements 1, 2, 3 connected to a, b, c, d through lines labeled makes.]
Multiplicity of E/R Relations

- one-one:

- many-one

- many-many
What does this say?
Attributes on Relationships

Person

name

address

Buys

What does this say?

date

Product

name

price
Multi-way Relationships

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

Can still model as a mathematical set (How?)

As a set of triples $\subseteq \text{Product} \times \text{Person} \times \text{Store}$
Q: What does the arrow mean?

A: Any person buys a given product from at most one store

[Fine print: Arrow pointing to E means that if we select one entity from each of the other entity sets in the relationship, those entities are related to at most one entity in E]
Q: What does the arrow mean?

A: Any person buys a given product from at most one store AND every store sells to every person at most one product
Converting Multi-way Relationships to Binary

Arrows go in which direction?
Converting Multi-way Relationships to Binary

Make sure you understand why!
3. Design Principles

What’s wrong?

Moral: Be faithful to the specifications of the application!
Design Principles: What’s Wrong?

Moral: pick the right kind of entities.
Design Principles: What’s Wrong?

Product
Purchase
Dates
Store
Person
date

Moral: don’t complicate life more than it already is.
From E/R Diagrams to Relational Schema

- Entity set $\rightarrow$ relation
- Relationship $\rightarrow$ relation
**Entity Set to Relation**

**Product**

<table>
<thead>
<tr>
<th>prod-ID</th>
<th>category</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo55</td>
<td>Camera</td>
<td>99.99</td>
</tr>
<tr>
<td>Pokenm19</td>
<td>Toy</td>
<td>29.99</td>
</tr>
</tbody>
</table>
N-N Relationships to Relations

Represent this in relations
Orders\( (\text{prod-ID}, \text{cust-ID}, \text{date}) \)
Shipment\( (\text{prod-ID}, \text{cust-ID}, \text{name}, \text{date}) \)
Shipping-Co\( (\text{name}, \text{address}) \)

<table>
<thead>
<tr>
<th>prod-ID</th>
<th>cust-ID</th>
<th>name</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo55</td>
<td>Joe12</td>
<td>UPS</td>
<td>4/10/2011</td>
</tr>
<tr>
<td>Gizmo55</td>
<td>Joe12</td>
<td>FEDEX</td>
<td>4/9/2011</td>
</tr>
</tbody>
</table>
N-1 Relationships to Relations

Represent this in relations
N-1 Relationships to Relations

Orders\((prod-ID, cust-ID, date1, name, date2)\)

Shipping-Co\((name, address)\)

Remember: no separate relations for many-one relationship
Multi-way Relationships to Relations

Product

- prod-ID
- price

Purchase

- Purchase \((prod-ID, ssn, name)\)

Person

- ssn
- name

Store

- name
- address
Modeling Subclasses

Some objects in a class may be special
- define a new class
- better: define a subclass

So --- we define subclasses in E/R
Subclasses

Product

name

category

price

isa isa

Software Product

Educational Product

platforms

Age Group

CSE 344 - 2019wi
Subclasses to Relations

Other ways to convert are possible

CSE 344 - 2019wi
Modeling Union Types 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 but imperfect (What’s wrong?)
Modeling Union Types with Subclasses

Solution 2: better, more laborious

- Person
- Company
- FurniturePiece
  - isa: Person
  - isa: Company
  - ownedBy: Owner
Weak Entity Sets

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

Team(sport, number, universityName)
University(name)
What Are the Keys of R?

A
B
R
T
C
V
D
E
Q
W
F
L
G
Z
K
S
U
V
H
Introduction to Data Management
CSE 344

Integrity Constraints
An integrity constraint is a condition specified on a database schema that restricts the data that can be stored in an instance of the database.

Why?

How?
Integrity Constraints Motivation

An integrity constraint is a condition specified on a database schema that restricts the data that can be stored in an instance of the database.

Why? Because we want application data to be consistent

How?
Integrity Constraints Motivation

An integrity constraint is a condition specified on a database schema that restricts the data that can be stored in an instance of the database.

Why? Because we want application data to be consistent

How? The DBMS checks and enforces IC during updates
Constraints in E/R Diagrams

• Keys

• Single-value constraints

• Referential integrity constraints

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

Underline:

- name
- category
- price
- ssn
- address
- name
- ssn
Single Value Constraints

makes

vs.

makes
Referential Integrity Constraints

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

Each product made by exactly one company.
Other Constraints

A Company entity is connected to at most 99 Product entities
Constraints in SQL

• Keys

• Attribute-level, tuple-level constraints

• General (complex) constraints

The more complex the constraint, the harder it is to check and to enforce
Key Constraints

Product(name, category)

CREATE TABLE Product (  
   name CHAR(30) PRIMARY KEY,  
   category VARCHAR(20))

OR:

CREATE TABLE Product (  
   name CHAR(30),  
   category VARCHAR(20),  
   PRIMARY KEY (name))
Keys with Multiple Attributes

Product(name, category, price)

CREATE TABLE Product (  
    name CHAR(30),  
    category VARCHAR(20),  
    price INT,  
    PRIMARY KEY (name, category))

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>10</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>20</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Photo</td>
<td>30</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>40</td>
</tr>
</tbody>
</table>
CREATE TABLE Product ( 
    productID CHAR(10),
    name CHAR(30),
    category VARCHAR(20),
    price INT,
    PRIMARY KEY (productID),
    UNIQUE (name, category))

There is at most one PRIMARY KEY; there can be many UNIQUE
Foreign Key Constraints

CREATE TABLE Purchase ( 
    prodName CHAR(30) 
    REFERENCES Product(name), 
    date DATETIME)

prodName is a foreign key to Product(name)
name must be a key in Product

Referential integrity constraints

May write just Product if name is PK
Foreign Key Constraints

- Example with multi-attribute primary key

```sql
CREATE TABLE Purchase (
    prodName CHAR(30),
    category VARCHAR(20),
    date DATETIME,
    FOREIGN KEY (prodName, category) REFERENCES Product(name, category)
)
```

- (name, category) must be a KEY in Product
What happens when data changes?

Types of updates:
- In Purchase: insert/update
- In Product: delete/update

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ProdName</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
</tbody>
</table>
What happens when data changes?

SQL policies for maintaining referential integrity:

- **NO ACTION** reject modifications (default)

- **CASCADE** after delete/update do delete/update

- **SET NULL** set foreign-key field to NULL

- **SET DEFAULT**

  CREATE TABLE ...  
  (pid int DEFAULT 42 REFERENCES...
Maintaining Referential Integrity

CREATE TABLE Purchase (  
  prodName CHAR(30),  
  category VARCHAR(20),  
  date DATETIME,  
  FOREIGN KEY (prodName, category)  
    REFERENCES Product(name, category)  
    ON UPDATE CASCADE  
    ON DELETE SET NULL  
)
Constraints on Attributes and Tuples

• Constraints on attributes:
  NOT NULL  -- obvious meaning...
  CHECK condition  -- any condition !

• Constraints on tuples
  CHECK condition
CREATE TABLE User (  
  uid int primary key,  
  firstName text,  
  lastName text NOT NULL,  
  age int CHECK (age > 12 and age < 120),  
  email text,  
  phone text,  
  CHECK (email is not NULL or phone is not NULL)  
)
Constraints on Attributes and Tuples

What does this constraint do?

What is the difference from Foreign-Key?

CREATE TABLE Purchase (prodName CHAR(30) CHECK (prodName IN (SELECT Product.name FROM Product)),
date DATETIME NOT NULL)
General Assertions

CREATE ASSERTION myAssert CHECK
(NOT EXISTS(
    SELECT Product.name
    FROM Product, Purchase
    WHERE Product.name = Purchase.prodName
    GROUP BY Product.name
    HAVING count(*) > 200)

But most DBMSs do not implement assertions
Because it is hard to support them efficiently
Instead, they provide triggers
Introduction to Data Management
CSE 344

Design Theory and BCNF
What makes good schemas?
Relational Schema Design

<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>

One person may have multiple phones, but lives in only one city.

Primary key is thus (SSN, PhoneNumber)

What is the problem with this schema?
Anomalies:
- **Redundancy** = repeat data
- **Update anomalies** = what if Fred moves to “Bellevue”? 
- **Deletion anomalies** = what if Joe deletes his phone number?

<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>
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>

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

How do we do this systematically?

• Start with some relational schema

• Find out its *functional dependencies* (FDs)

• Use FDs to *normalize* the relational schema
Functional Dependencies (FDs)

**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:

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

**A_1 \ldots A_n determines B_1 \ldots B_m**
**Functional Dependencies (FDs)**

**Definition** \( A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n \) holds in \( R \) if:

\[
\forall t, t' \in R, \quad (t.A_1 = t'.A_1 \land \ldots \land t.A_m = t'.A_m \rightarrow t.B_1 = t'.B_1 \land \ldots \land t.B_n = t'.B_n)
\]

<table>
<thead>
<tr>
<th>( R )</th>
<th>( A_1 )</th>
<th>( \ldots )</th>
<th>( A_m )</th>
<th>( B_1 )</th>
<th>( \ldots )</th>
<th>( B_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t' )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If \( t, t' \) agree here then \( t, t' \) agree here.
Example

An FD holds, or does not hold on an instance:

<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>E3542</td>
<td>Mike</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 not Phone → 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>E3542</td>
<td>Mike</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 → Phone
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>E3542</td>
<td>Mike</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>

But not Phone  →  Position
### 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>Tweaker</td>
<td>Gadget</td>
<td>Red</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Stationary</td>
<td>Green</td>
<td>Office-supp.</td>
<td>59</td>
</tr>
</tbody>
</table>

Which FD’s hold?

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

- FD holds or does not hold on an instance

- If we can be sure that every instance of $R$ will be one in which a given FD is true, then we say that $R$ satisfies the FD

- If we say that $R$ satisfies an FD, we are stating a constraint on $R$
Why bother with FDs?

<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>

Anomalies:

- **Redundancy** = repeat data
- **Update anomalies** = what if Fred moves to “Bellevue”?
- **Deletion anomalies** = what if Joe deletes his phone number?
An Interesting Observation

If all these FDs are true:

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

Then this FD also holds:

- name, category → price

If we find out from application domain that a relation satisfies some FDs, it doesn’t mean that we found all the FDs that it satisfies! There could be more FDs implied by the ones we have.
Given a set of attributes $A_1, \ldots, A_n$

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

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

Closures:

$name^+ = \{\text{name, color}\}$

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

$\text{color}^+ = \{\text{color}\}$
Closure Algorithm

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

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:

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

$\{\text{name, category}\}^+ =$

$\{\text{name, category, color, department, price}\}$

Hence: name, category $\rightarrow$ color, department, price
Why do we care?

• The closure allows us to compute all FDs implied by a given FD; Here is how:

• To check if the FD implies $A \rightarrow B$
  – Compute $A^+$
  – Check if $B \subseteq A^+$
Example

In class:

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

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

Compute \( \{A,B\}^+ \) \hspace{1cm} X = \{A, B, \}

Compute \( \{A, F\}^+ \) \hspace{1cm} X = \{A, F, \}
Example

In class:

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

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

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

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

In class:

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

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

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

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

In class:

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

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

Compute \( \{A, B\}^+ \) \hspace{1cm} X = \{A, B, C, D, E\}

Compute \( \{A, F\}^+ \) \hspace{1cm} X = \{A, F, B, C, D, E\}

What is the key of \( R \)?
Find all FD’s implied by:

A, B $\rightarrow$ C
A, D $\rightarrow$ B
B $\rightarrow$ D
Practice at Home

Find all FD’s implied by:

A, B → C
A, D → B
B → D

Step 1: Compute X⁺, for every X:

A⁺ = A, B⁺ = BD, C⁺ = C, D⁺ = D
AB⁺ = ABCD, AC⁺ = AC, AD⁺ = ABCD,
   BC⁺ = BCD, BD⁺ = BD, CD⁺ = CD
ABC⁺ = ABD⁺ = ACD⁺ = ABCD (no need to compute—why?)
BCD⁺ = BCD, ABCD⁺ = ABCD
Practice at Home

Find all FD’s implied by:

- A, B → C
- A, D → B
- B → D

Step 1: Compute $X^+$, for every $X$:

- $A^+ = A$
- $B^+ = BD$
- $C^+ = C$
- $D^+ = D$

$AB^+ = ABCD$, $AC^+ = AC$, $AD^+ = ABCD$, $BC^+ = BCD$, $BD^+ = BD$, $CD^+ = CD$

$ABC^+ = ABD^+ = ACD^+ = ABCD$ (no need to compute– why?)

$BCD^+ = BCD$, $ABCD^+ = ABCD$

Step 2: Enumerate all FD’s $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

- $AB \rightarrow CD$, $AD \rightarrow BC$, $ABC \rightarrow D$, $ABD \rightarrow C$, $ACD \rightarrow B$
Keys

• A **superkey** is a set of attributes $A_1, \ldots, A_n$ s.t. for any other attribute $B$, we have $A_1, \ldots, A_n \rightarrow B$

• A **key** is a minimal superkey
  – A superkey and for which no subset is a superkey
Computing (Super)Keys

- For all sets $X$, compute $X^+$
- If $X^+ = [\text{all attributes}]$, then $X$ is a superkey
- Try reducing to the minimal $X$’s to get the key
Example

Product(name, price, category, color)

name, category $\rightarrow$ price
category $\rightarrow$ color

What is the key?
What is the key?

(name, category) + = { name, category, price, color }

Hence (name, category) is a key
Key or Keys?

We can we have more than one key!

What are the keys here?

A → B
B → C
C → A
Key or Keys?

We can have more than one key!

What are the keys here?

A → B
B → C
C → A

AB → C
BC → A
Key or Keys?

We can have more than one key!

What are the keys here?

A → B
B → C
C → A

A → BC
B → AC
Eliminating Anomalies

Main idea:

• $X \rightarrow A$ is OK if $X$ is a (super)key

• $X \rightarrow A$ is not OK otherwise
  – Need to decompose the table, but how?
Boyce-Codd Normal Form

Dr. Raymond F. Boyce
Edgar Frank “Ted” Codd

"A Relational Model of Data for Large Shared Data Banks"
There are no “bad” FDs:

**Definition.** A relation R is in BCNF if:

Whenever \( X \rightarrow B \) is a non-trivial dependency, then \( X \) is a superkey.

Equivalently:

**Definition.** A relation R is in BCNF if:

\[ \forall X, \text{ either } X^+ = X \text{ (i.e., } X \text{ is not in any FDs) } \text{ or } X^+ = [\text{all attributes}] \text{ (computed using FDs)} \]
BCNF Decomposition Algorithm

Normalize(R)
find X s.t.: $X \neq X^+$ and $X^+ \neq$ [all attributes]
if (not found) then “R is in BCNF”
let $Y = X^+ - X$; $Z = [all\ attributes] - X^+$
decompose R into $R_1(X \cup Y)$ and $R_2(X \cup Z)$
Normalize($R_1$); Normalize($R_2$);
### 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>

The only key is: \{SSN, PhoneNumber\}

Hence \textbf{SSN} \rightarrow \textbf{Name, City} is a “bad” dependency

In other words:
\textbf{SSN⁺} = \textbf{SSN, Name, City} and is neither \textbf{SSN} nor \textbf{All Attributes}
Example BCNF Decomposition

<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>

SSN → Name, City

Let’s check anomalies:
- Redundancy?
- Update?
- Delete?
Example BCNF Decomposition

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

SSN $\rightarrow$ name, age

age $\rightarrow$ hairColor

Find $X$ s.t.: $X \neq X^+$ and $X^+ \neq \{\text{all attributes}\}$
Example BCNF Decomposition

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

SSN → name, age
age → hairColor

Iteration 1: Person: SSN+ = SSN, name, age, hairColor
Decompose into: P(SSN, name, age, hairColor)
Phone(SSN, phoneNumber)
Find $X$ s.t.: $X \neq X^+$ and $X^+ \neq \text{[all attributes]}$

**Example BCNF Decomposition**

$\text{Person(name, SSN, age, hairColor, phoneNumber)}$

- $\text{SSN} \rightarrow \text{name, age}$
- $\text{age} \rightarrow \text{hairColor}$

**Iteration 1:**
- **Person:** $\text{SSN}^+ = \text{SSN, name, age, hairColor}$
- Decompose into: $P(\text{SSN, name, age, hairColor})$
  - $\text{Phone(SSN, phoneNumber)}$

**Iteration 2:**
- **P:** $\text{age}^+ = \text{age, hairColor}$
- Decompose: $\text{People(SSN, name, age)}$
  - $\text{Hair(age, hairColor)}$
  - $\text{Phone(SSN, phoneNumber)}$

What are the keys?
Example BCNF Decomposition

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

SSN \rightarrow name, age
age \rightarrow hairColor

Iteration 1: Person: SSN+ = SSN, name, age, hairColor
Decompose into: P(SSN, name, age, hairColor)
Phone(SSN, phoneNumber)

Iteration 2: P: age+ = age, hairColor
Decompose: People(SSN, name, age)
Hair(age, hairColor)
Phone(SSN, phoneNumber)

Find X s.t.: X \neq X^+ and X^+ \neq \{all attributes\}
Example: BCNF
Example: BCNF

Recall: find $X$ s.t. $X \subset X^+ \subset \text{[all-attrs]}$
Example: BCNF

R(A,B,C,D)

\[ A^+ = ABC \neq ABCD \]
Example: BCNF

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

$A + = ABC \neq ABCD$

$R_1(A,B,C)$

$R_2(A,D)$

$A \rightarrow B$

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

Example: BCNF

R(A,B,C,D)
A⁺ = ABC ≠ ABCD

R₁(A,B,C)
B⁺ = BC ≠ ABC

R₂(A,D)
Example: BCNF

R(A,B,C,D)

A⁺ = ABC ≠ ABCD

R₁(A,B,C)

B⁺ = BC ≠ ABC

R₁₁(B,C)

R₁₂(A,B)

R₂(A,D)

What are the keys?

What happens if in R we first pick B⁺? Or AB⁺?
Decompositions in General

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

\[ S_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \quad S_2(A_1, \ldots, A_n, C_1, \ldots, C_p) \]

\[ S_1 = \text{projection of } R \text{ on } A_1, \ldots, A_n, B_1, \ldots, B_m \]
\[ S_2 = \text{projection of } R \text{ on } A_1, \ldots, A_n, C_1, \ldots, C_p \]
Lossless Decomposition

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

- **Name**
  - Gizmo
  - OneClick
  - Gizmo

- **Price**
  - 19.99
  - 24.99
  - 19.99

- **Category**
  - Gadget
  - Camera

CSE 344 - 2019wi
Lossy Decomposition

What is lossy here?

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Camera</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>
Lossy Decomposition

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Camera</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>
# Lossy Decomposition

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

- **Table 1:**
<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Camera</td>
</tr>
</tbody>
</table>

- **Table 2:**
<table>
<thead>
<tr>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>
Decomposition in General

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

\[ S_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \]

\[ S_2(A_1, \ldots, A_n, C_1, \ldots, C_p) \]

Let:

\[ S_1 = \text{projection of } R \text{ on } A_1, \ldots, A_n, B_1, \ldots, B_m \]

\[ S_2 = \text{projection of } R \text{ on } A_1, \ldots, A_n, C_1, \ldots, C_p \]

The decomposition is called **lossless** if \( R = S_1 \bowtie S_2 \)

**Fact:** If \( A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m \) then the decomposition is lossless

It follows that every BCNF decomposition is lossless
Testing for Lossless Join

If we decompose $R$ into $\Pi_{S_1}(R)$, $\Pi_{S_2}(R)$, $\Pi_{S_3}(R)$, …
Is it true that $S_1 \bowtie S_2 \bowtie S_3 \bowtie … = R$?

To check “=” we need to check “⊆” and “⊇”

$R \subseteq S_1 \bowtie S_2 \bowtie S_3 \bowtie …$ always holds (why?)

$R \supseteq S_1 \bowtie S_2 \bowtie S_3 \bowtie …$ neet to check
The Chase Test for Lossless Join

$R(A,B,C,D) = S_1(A,D) \bowtie S_2(A,C) \bowtie S_3(B,C,D)$

$R$ satisfies: $A \rightarrow B$, $B \rightarrow C$, $CD \rightarrow A$

$S_1 = \Pi_{AD}(R)$, $S_2 = \Pi_{AC}(R)$, $S_3 = \Pi_{BCD}(R)$
The Chase Test for Lossless Join

R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D)
R satisfies: A \rightarrow B, B \rightarrow C, CD \rightarrow A

S1 = \Pi_{AD}(R), S2 = \Pi_{AC}(R), S3 = \Pi_{BCD}(R)
R \subseteq S1 \bowtie S2 \bowtie S3
To check: R \supseteq S1 \bowtie S2 \bowtie S3
The Chase Test for Lossless Join

Example from textbook Ch. 3.4.2

\[ R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D) \]

R satisfies: \( A \rightarrow B, B \rightarrow C, CD \rightarrow A \)

\[ S1 = \Pi_{AD}(R), S2 = \Pi_{AC}(R), S3 = \Pi_{BCD}(R) \]

\( R \subseteq S1 \bowtie S2 \bowtie S3 \)

To check: \( R \supseteq S1 \bowtie S2 \bowtie S3 \)

Suppose \((a,b,c,d) \in S1 \bowtie S2 \bowtie S3\) Is it also in R?

Lossless?
The Chase Test for Lossless Join

\[ R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D) \]

R satisfies: \( A \rightarrow B \), \( B \rightarrow C \), \( CD \rightarrow A \)

\( S1 = \Pi_{AD}(R) \), \( S2 = \Pi_{AC}(R) \), \( S3 = \Pi_{BCD}(R) \)

\( R \subseteq S1 \bowtie S2 \bowtie S3 \)

To check: \( R \supseteq S1 \bowtie S2 \bowtie S3 \)

Suppose \((a,b,c,d) \in S1 \bowtie S2 \bowtie S3\) Is it also in \( R \)?

R must contain the following tuples:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b1</td>
<td>c1</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

Why?

\((a,d) \in S1 = \Pi_{AD}(R)\)
The Chase Test for Lossless Join

Example from textbook Ch. 3.4.2

\[ R(A,B,C,D) = S_1(A,D) \bowtie S_2(A,C) \bowtie S_3(B,C,D) \]

\( R \) satisfies: \( A \rightarrow B, B \rightarrow C, CD \rightarrow A \)

\( S_1 = \Pi_{AD}(R), S_2 = \Pi_{AC}(R), S_3 = \Pi_{BCD}(R) \)

\( R \subseteq S_1 \bowtie S_2 \bowtie S_3 \)

To check: \( R \supseteq S_1 \bowtie S_2 \bowtie S_3 \)

Suppose \((a,b,c,d) \in S_1 \bowtie S_2 \bowtie S_3\) Is it also in \( R \)?

\( R \) must contain the following tuples:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b1</td>
<td>c1</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>b2</td>
<td>c</td>
<td>d2</td>
</tr>
</tbody>
</table>

Why?

\((a,d) \in S_1 = \Pi_{AD}(R)\)

\((a,c) \in S_2 = \Pi_{BD}(R)\)
The Chase Test for Lossless Join

Example from textbook Ch. 3.4.2

\[ R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D) \]

\( R \) satisfies: \( A \rightarrow B, B \rightarrow C, CD \rightarrow A \)

\( S1 = \Pi_{AD}(R), S2 = \Pi_{AC}(R), S3 = \Pi_{BCD}(R) \)

\( R \subseteq S1 \bowtie S2 \bowtie S3 \)

To check: \( R \supseteq S1 \bowtie S2 \bowtie S3 \)

Suppose \( (a,b,c,d) \in S1 \bowtie S2 \bowtie S3 \) Is it also in \( R \)?

\( R \) must contain the following tuples:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b1</td>
<td>c1</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>b2</td>
<td>c</td>
<td>d2</td>
</tr>
<tr>
<td>a3</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

Why?

\( (a,d) \in S1 = \Pi_{AD}(R) \)
\( (a,c) \in S2 = \Pi_{BD}(R) \)
\( (b,c,d) \in S3 = \Pi_{BCD}(R) \)
The Chase Test for Lossless Join

Example from textbook Ch. 3.4.2

\[ R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D) \]

R satisfies: \( A \rightarrow B, B \rightarrow C, CD \rightarrow A \)

\[ S1 = \Pi_{AD}(R), S2 = \Pi_{AC}(R), S3 = \Pi_{BCD}(R) \]

\[ R \subseteq S1 \bowtie S2 \bowtie S3 \]

To check: \( R \supseteq S1 \bowtie S2 \bowtie S3 \)

Suppose \((a,b,c,d) \in S1 \bowtie S2 \bowtie S3\) Is it also in R?

R must contain the following tuples:

"Chase" them (apply FDs):

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b1</td>
<td>c1</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>b2</td>
<td>c</td>
<td>d2</td>
</tr>
<tr>
<td>a3</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

Why?

(a,d) \in S1 = \Pi_{AD}(R)

(a,c) \in S2 = \Pi_{BD}(R)

(b,c,d) \in S3 = \Pi_{BCD}(R)
The Chase Test for Lossless Join

\[ R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D) \]
R satisfies: \( A \rightarrow B, B \rightarrow C, CD \rightarrow A \)

\[ R \subseteq S1 \bowtie S2 \bowtie S3 \]
To check: \( R \supseteq S1 \bowtie S2 \bowtie S3 \)
Suppose \( (a,b,c,d) \in S1 \bowtie S2 \bowtie S3 \) Is it also in \( R \)?
R must contain the following tuples:

"Chase" them (apply FDs):

\[ \begin{array}{cccc}
A & B & C & D \\
\hline
a & b1 & c1 & d \\
a & b1 & c & d2 \\
a3 & b & c & d
\end{array} \]

\[ \begin{array}{cccc}
A & B & C & D \\
\hline
a & b1 & c & d \\
a & b1 & c & d2 \\
a3 & b & c & d
\end{array} \]

\[ \begin{array}{cccc}
A & B & C & D \\
\hline
(a,d) \in S1 = \Pi_{AD}(R) \\
(a,b2,c,d2) \in S2 = \Pi_{BD}(R) \\
(b,c,d) \in S3 = \Pi_{BCD}(R)
\end{array} \]
The Chase Test for Lossless Join

Example from textbook Ch. 3.4.2

\[ R(A, B, C, D) = S1(A, D) \bowtie S2(A, C) \bowtie S3(B, C, D) \]

\( R \) satisfies: \( A \rightarrow B, B \rightarrow C, CD \rightarrow A \)

\( S1 = \Pi_{AD}(R), S2 = \Pi_{AC}(R), S3 = \Pi_{BCD}(R) \)

\( R \subseteq S1 \bowtie S2 \bowtie S3 \)

To check: \( R \supseteq S1 \bowtie S2 \bowtie S3 \)

Suppose \( (a, b, c, d) \in S1 \bowtie S2 \bowtie S3 \) Is it also in \( R \)?

\( R \) must contain the following tuples:

"Chase" them (apply FDs):

\[ A \rightarrow B \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b1</td>
<td>c1</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>b1</td>
<td>c</td>
<td>d2</td>
</tr>
<tr>
<td>a3</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

\[ B \rightarrow C \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b1</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>b1</td>
<td>c</td>
<td>d2</td>
</tr>
<tr>
<td>a3</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

\[ CD \rightarrow A \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b1</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>b1</td>
<td>c</td>
<td>d2</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

Hence \( R \) contains \( (a, b, c, d) \)

Lossless?
The Chase Test for Lossless Join

Example from textbook Ch. 3.4.2

\[ R(A,B,C,D) = S_1(A,D) \bowtie S_2(A,C) \bowtie S_3(B,C,D) \]

R satisfies: \( A \rightarrow B \), \( B \rightarrow C \), \( CD \rightarrow A \)

S1 = \( \Pi_{AD}(R) \), S2 = \( \Pi_{AC}(R) \), S3 = \( \Pi_{BCD}(R) \)

R \subseteq S1 \bowtie S2 \bowtie S3

To check: R \supseteq S1 \bowtie S2 \bowtie S3

Suppose \((a,b,c,d) \in S1 \bowtie S2 \bowtie S3\) Is it also in R?

R must contain the following tuples:

\begin{align*}
A & \rightarrow B \\
B & \rightarrow C \\
CD & \rightarrow A
\end{align*}

“Chase” them (apply FDs):

\begin{align*}
& A \rightarrow B \\
& a \rightarrow b1 \\
& a \rightarrow b1 \\
& a3 \rightarrow b \\
& a \rightarrow \text{red}
\end{align*}

\begin{align*}
& B \rightarrow C \\
& b1 \rightarrow c1 \\
& b1 \rightarrow c \\
& b \rightarrow c \\
& b1 \rightarrow \text{red}
\end{align*}

\begin{align*}
& CD \rightarrow A \\
& c1 \rightarrow d \\
& c \rightarrow d \\
& c \rightarrow d \\
& c \rightarrow \text{red}
\end{align*}

Why?

\begin{align*}
(a,d) & \in S1 = \Pi_{AD}(R) \\
(a,c) & \in S2 = \Pi_{BD}(R) \\
(b,c,d) & \in S3 = \Pi_{BCD}(R)
\end{align*}

Hence R contains \((a,b,c,d)\)

Lossless? YES!
Schema Refinements

= Normal Forms

- 1st Normal Form = all tables are flat
- 2nd Normal Form = obsolete
- Boyce Codd Normal Form = no bad FDs
- 3rd Normal Form = see book
  - BCNF removes anomalies, but may lose some FDs
    (see book 3.4.4)
  - 3NF preserves all FD’s, but may still have some anomalies
Conclusion

• E/R diagrams are means to structurally visualize and design relational schemas

• Normalization is a principled way of converting schemas into a form that avoid such redundancies.

• BCNF and 3NF are the most widely used normalized form in practice