CSE544
Data Management

Lecture 3
Schema Normalization
Announcements

- Monday: no class (MLK day)
- Tuesday: project groups due
- Wednesday: first review due
- Next Saturday: homework 1 due
  - git pull # just in case
  - git commit --a --m 'your message here'
  - git push
Database Design

• The relational model is great, but how do I design my database schema?
Outline

• Conceptual db design: entity-relationship model

• Problematic database designs

• Functional dependencies

• Normal forms and schema normalization
Conceptual Schema Design

Conceptual Model:

Relational Model: plus FD’s (FD = functional dependency)

Normalization: Eliminates anomalies
Entity-Relationship Diagram

Attributes
- name

Entity sets
- Patient

Relationship sets
- patient_of
Entity-Relationship Diagram

Attributes
- name

Entity sets
- Patient

Relationship sets
- patient_of

Doctor

Patient
Entity-Relationship Diagram

**Patient**
- **Attributes**
  - name
  - zip
- **Entity sets**

**Doctor**
- **Attributes**
  - dno
  - specialty
  - name
- **Entity sets**

**Relationship sets**
- patient_of
Entity-Relationship Diagram

Patient

- name
- pno
- zip

Doctor

- dno
- specialty
- name

Attributes
- name

Entity sets
- Patient

Relationship sets
- patient_of
Entity-Relationship Relationship Diagram

Attributes
- name

Entity sets
- Patient

Relationship sets
- patient_of

Entity sets
- Patient

Relationship sets
- patient_of
Entity-Relationship Model

- Typically, each entity has a key
- ER relationships can include multiplicity
  - One-to-one, one-to-many, etc.
  - Indicated with arrows
- Can model multi-way relationships
- Can model subclasses
- And more...
E/R To Relations

**Patient**
- **pno**: P311
- **name**: Alice
- **zip**: 98765

**Doctor**
- **dno**: D007
- **name**: Bob
- **spec**: cardio

**Patient_of**
- **pno**: P311
- **dno**: D007
- **since**: 2001

**Patient**

<table>
<thead>
<tr>
<th>pno</th>
<th>name</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>P311</td>
<td>Alice</td>
<td>98765</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Patient_of**

<table>
<thead>
<tr>
<th>pno</th>
<th>dno</th>
<th>since</th>
</tr>
</thead>
<tbody>
<tr>
<td>P311</td>
<td>D007</td>
<td>2001</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Doctor**

<table>
<thead>
<tr>
<th>dno</th>
<th>name</th>
<th>spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>D007</td>
<td>Bob</td>
<td>cardio</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Notice Many-One Relationship

Patient

<table>
<thead>
<tr>
<th>pno</th>
<th>name</th>
<th>zip</th>
<th>dno</th>
<th>since</th>
</tr>
</thead>
<tbody>
<tr>
<td>P311</td>
<td>Alice</td>
<td>98765</td>
<td>D007</td>
<td>2001</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Doctor

<table>
<thead>
<tr>
<th>dno</th>
<th>name</th>
<th>spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>D007</td>
<td>Bob</td>
<td>cardio</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Subclasses to Relations

Product

- isa - Software Product
  - platforms
- isa - Educational Product
  - Age Group

- name
- category
- price
Subclasses to Relations

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>99</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>49</td>
<td>photo</td>
</tr>
<tr>
<td>Toy</td>
<td>39</td>
<td>gadget</td>
</tr>
</tbody>
</table>
Subclasses to Relations

<table>
<thead>
<tr>
<th>Product</th>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>99</td>
<td>gadget</td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>49</td>
<td>photo</td>
<td></td>
</tr>
<tr>
<td>Toy</td>
<td>39</td>
<td>gadget</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sw.Product</th>
<th>Name</th>
<th>platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td></td>
<td>unix</td>
</tr>
</tbody>
</table>
Subclasses to Relations

Product

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>99</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>49</td>
<td>photo</td>
</tr>
<tr>
<td>Toy</td>
<td>39</td>
<td>gadget</td>
</tr>
</tbody>
</table>

Software Product

<table>
<thead>
<tr>
<th>Name</th>
<th>platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>unix</td>
</tr>
</tbody>
</table>

Educational Product

<table>
<thead>
<tr>
<th>Name</th>
<th>Age Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>toddler</td>
</tr>
<tr>
<td>Toy</td>
<td>senior</td>
</tr>
</tbody>
</table>
E/R Diagram to Relations

• Each entity set becomes a relation with a key

• Each relationship set becomes a relation with foreign keys except many-one relationships: just add a fk

• Each isA relationship becomes another relation, with both a key and foreign key
Outline

• Conceptual db design: entity-relationship model

• Problematic database designs

• Functional dependencies

• Normal forms and schema normalization
## 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?
Relational Schema Design

Anomalies:

- **Redundancy** = repeat data for Fred
- **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:

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

EmplID → 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 \(\rightarrow\) 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>Green</td>
<td>Toys</td>
<td>99</td>
</tr>
</tbody>
</table>

Which FD’s hold?

color, category → department
name → color

name → color

category → department

color, category → 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 $R$
An Interesting Observation

If all these FDs are true:
- name $\rightarrow$ color
- category $\rightarrow$ department
- color, category $\rightarrow$ price

Then this FD also holds:
- name, category $\rightarrow$ price

Find out from application domain some FDs, Compute all FD’s implied by them
Closure of a set of Attributes

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

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

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

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

**Example:**
1. name $\rightarrow$ color
2. category $\rightarrow$ department
3. color, category $\rightarrow$ price
Closure of a set of Attributes

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

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

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

**Closures:**

$name^+ = \{\text{name, color}\}$
Closure of a set of Attributes

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

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

**Example:**

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

**Closures:**

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

$$\{\text{name, category}\}^+ = \{\text{name, category, color, department, price}\}$$
Given a set of attributes $A_1, \ldots, A_n$

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

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

Closures:
- $name^+ = \{name, color\}$
- $\{name, category\}^+ = \{name, category, color, department, price\}$
- $color^+ = \{color\}$
Keys

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

- A key is a minimal superkey (no subset is a superkey)
Computing (Super)Keys

• For all sets $X$, compute $X^+$

• If $X^+ = \{\text{all attributes}\}$, then $X$ is a superkey

• If, in addition, no subset of $X$ is a superkey, then $X$ is a key
Example

Product(name, price, category, color)

name, category → price
category → color

What is the key?
Example

Product(name, price, category, color)

What is the key?

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

Product(name, price, category, color)

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

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

Can we have more than one key?
Key or Keys?

Can we have more than one key?

A → B
B → C
C → A

what are the keys here?
Key or Keys?

Can we have more than one key?

A $\rightarrow$ B
B $\rightarrow$ C
C $\rightarrow$ A

what are the keys here?

AB $\rightarrow$ C
BC $\rightarrow$ A
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
Boyce-Codd Normal Form

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:

$\forall X$, either $X^+ = X$ or $X^+ = \text{[all attributes]}$
BCNF Decomposition Algorithm

Normalize(R)

find X s.t.: X ≠ X⁺ ≠ [all attributes]

if (not found) then “R is in BCNF”

let Z = [all attributes] - X⁺

decompose R into R1(X⁺) and R2(X ∪ Z)

Normalize(R1); Normalize(R2);
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 **SSN \rightarrow Name, City** is a “bad” dependency

In other words:
**SSN+ = SSN, Name, City** and is neither SSN nor All Attributes
Example BCNF Decomposition

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

SSN → name, age
age → hairColor

Find X s.t.: X ≠ X⁺ and X⁺ ≠ [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)
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)

What are the keys?

Find X s.t.: $X \neq X^+$ and $X^+ \neq$ [all attributes]
Find X s.t.: X $\neq X^+$ and $X^+ \neq \text{[all attributes]}$

**Example BCNF Decomposition**

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

<table>
<thead>
<tr>
<th>SSN $\rightarrow$ name, age</th>
</tr>
</thead>
<tbody>
<tr>
<td>age $\rightarrow$ hairColor</td>
</tr>
</tbody>
</table>

**Iteration 1:**

Person: \( SSN^+ = SSN, \text{name, age, hairColor} \)

Decompose into:

\( P(\text{SSN, name, age, hairColor}) \)

\( \text{Phone(SSN, phoneNumber)} \)

**Iteration 2:**

\( P: \text{age}^+ = \text{age, hairColor} \)

Decompose:

\( \text{People(SSN, name, age)} \)

\( \text{Hair(age, hairColor)} \)

\( \text{Phone(SSN, phoneNumber)} \)

Note the keys!
Example: BCNF

A → B
B → C

R(A, B, C, D)

R(A, B, C, D)
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 ≠ ABCD

R₁(A,B,C)

R₂(A,D)

A → B
B → C
Example: BCNF

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

$A^+ = ABC \neq ABCD$

$R_1(A,B,C)$

$B^+ = BC \neq ABC$

$R_2(A,D)$

$A \rightarrow B$

$B \rightarrow C$
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

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

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

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

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

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

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

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

Name | Category
--- | ---
Gizmo | Gadget
OneClick | Camera
Gizmo | Camera

Price | Category
--- | ---
19.99 | Gadget
24.99 | Camera
19.99 | Camera
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>
Decomposition in General

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

Let:

- \( S_1 = \text{projection of } R \text{ on } A_1, ..., A_n, B_1, ..., B_m \)
- \( S_2 = \text{projection of } R \text{ on } A_1, ..., A_n, C_1, ..., C_p \)

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

Fact: If \( A_1, ..., A_n \rightarrow B_1, ..., B_m \) 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 $S_1 \bowtie S_2 \bowtie S_3 \bowtie \cdots = R$?

That is true if we can show that:

$R \subseteq S_1 \bowtie S_2 \bowtie S_3 \bowtie \cdots$ but this always holds; why?

$R \supseteq S_1 \bowtie S_2 \bowtie S_3 \bowtie \cdots$ neet to check
The Chase Test

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

\[ R \text{ satisfies: } A \rightarrow B, \ B \rightarrow C, \ CD \rightarrow A \]

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

\[ \text{hence } R \subseteq S_1 \bowtie S_2 \bowtie S_3 \]

\[ \text{Need to check: } R \supseteq S_1 \bowtie S_2 \bowtie S_3 \]
The Chase Test

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

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

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

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

Need 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 \( R \)?
The Chase Test

R(A,B,C,D) = S1(A,D) ⋈ S2(A,C) ⋈ S3(B,C,D)
R satisfies: A→B, B→C, CD→A

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

Need to check: R \supseteq S1 ⋈ S2 ⋈ S3
Suppose (a,b,c,d) \in S1 ⋈ S2 ⋈ 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>
</tbody>
</table>

Why ?
(a,d) \in S1 = \Pi_{AD}(R)
**The Chase Test**

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

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

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

Need 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>a</td>
<td>b1</td>
<td>c1</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>b2</td>
<td>c</td>
<td>d2</td>
</tr>
</tbody>
</table>

Why?

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

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

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

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

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

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

Need 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></td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>b2</td>
<td></td>
<td>c</td>
<td>d2</td>
</tr>
<tr>
<td>a3</td>
<td></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

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),
hence R \subseteq S1 \bowtie S2 \bowtie S3

Need 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></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>
<tr>
<td>a</td>
<td>b2</td>
<td>c</td>
<td>d2</td>
<td></td>
</tr>
<tr>
<td>a3</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></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

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

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

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

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

Need 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>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>

<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>d</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

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

\[ R \text{ satisfies: } A \rightarrow B, B \rightarrow C, CD \rightarrow A \]

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

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

Need 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 \(R\)?

\[ R \text{ must contain the following tuples:} \]

"Chase" them (apply FDs):

\[
\begin{array}{cccc}
A & B & C & D \\
\text{a} & b_1 & c_1 & d \\
\text{a} & b_1 & c & d_2 \\
\text{a} & b_1 & c & d \\
\text{a} & b & c & d \\
\end{array}
\]

Why?

\[
\begin{array}{cccc}
A & B & C & D \\
\text{(a,d)} \in S_1 = \Pi_{AD}(R) \\
\text{(a,c)} \in S_2 = \Pi_{BD}(R) \\
(b,c,d) \in S_3 = \Pi_{BCD}(R) \\
\end{array}
\]

Hence \( R \) contains \((a,b,c,d)\)
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 is lossless but can cause loss of ability to check some FDs
  - 3NF fixes that (is lossless and dependency-preserving), but some tables might not be in BCNF – i.e., they may have redundancy anomalies