Lectures 8 and 9: Database Design

Wednesday & Friday, April 10 & 12
Announcements/Reminders

- Homework 1: solutions are posted
- Homework 2: posted (due Friday, April 20)
- Project Phase 1 due Friday, April 12
Outline

• The relational data model: 3.1
• Functional dependencies: 3.4
Schema Refinements = Normal Forms

• 1st Normal Form = all tables are flat
• 2nd Normal Form = obsolete
• Boyce Codd Normal Form = will study
• 3rd Normal Form = see book
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, DB, OS</td>
</tr>
<tr>
<td>Bob</td>
<td>3.7</td>
<td>DB, OS</td>
</tr>
<tr>
<td>Carol</td>
<td>3.9</td>
<td>Math, 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>
Relational Schema Design

Conceptual Model:

- **Product**: name, price
- **Person**: name, ssn
- **buys** relationship

Relational Model: plus FD’s

Normalization: Eliminates *anomalies*
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
Relational Schema Design

Recall set attributes (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>

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

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>

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

• Start with some relational schema
• Find out its functional dependencies
• Use them to design a better relational schema
Functional Dependencies

• A form of constraint
  – hence, part of the schema
• Finding them is part of the database design
• Also used in normalizing the relations
Functional Dependencies

Definition:

If two tuples agree on the attributes

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

then they must also agree on the attributes

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

Formally:

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]
When Does an FD Hold

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

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

\( \text{R} \)

<table>
<thead>
<tr>
<th></th>
<th>( A_1 )</th>
<th>( \ldots )</th>
<th>( A_m )</th>
<th>( B_1 )</th>
<th>( \ldots )</th>
<th>( B_m )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\text{t}</td>
<td>\text{t}</td>
<td>\text{t}</td>
<td>\text{t}</td>
<td>\text{t}</td>
<td>\text{t}</td>
</tr>
<tr>
<td>( t' )</td>
<td>\text{t}</td>
<td>\text{t}</td>
<td>\text{t}</td>
<td>\text{t}</td>
<td>\text{t}</td>
<td>\text{t}</td>
</tr>
</tbody>
</table>

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

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 $\rightarrow$ 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 $\rightarrow$ Position
Example

FD’s are constraints:
• 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

<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>Black</td>
<td>Toys</td>
<td>99</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Stationary</td>
<td>Green</td>
<td>Office-supp.</td>
<td>59</td>
</tr>
</tbody>
</table>

What about this one?
An Interesting Observation

If all these FDs are true:

name → color
category → department
color, category → price

Then this FD also holds:

name, category → price

Why ??
Goal: Find ALL Functional Dependencies

• Anomalies occur when certain “bad” FDs hold

• We know some of the FDs

• Need to find all FDs, then look for the bad ones
Armstrong’s Rules (1/3)

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

Is equivalent to

\[ A_1, A_2, \ldots, A_n \rightarrow B_1 \]
\[ A_1, A_2, \ldots, A_n \rightarrow B_2 \]
\[ \ldots \]
\[ A_1, A_2, \ldots, A_n \rightarrow B_m \]
Armstrong’s Rules (1/3)

Trivial Rule

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

where $i = 1, 2, \ldots, n$

Why?
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 \( A_1, A_2, \ldots, A_n \rightarrow C_1, C_2, \ldots, C_p \)

Why?
<table>
<thead>
<tr>
<th></th>
<th>A₁</th>
<th>...</th>
<th>Aₘ</th>
<th>B₁</th>
<th>...</th>
<th>Bₘ</th>
<th>C₁</th>
<th>...</th>
<th>Cₚ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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>

THIS IS TOO HARD ! Let’s see an easier way.
Closure of a set of Attributes

**Given** a set of attributes \( A_1, \ldots, A_n \)

The **closure** \( \{A_1, \ldots, A_n\}^+ \) = the set of attributes \( B \)

s.t. \( A_1, \ldots, A_n \rightarrow B \)

Example:

Closures:

- name \( \rightarrow \) color
- category \( \rightarrow \) department
- color, category \( \rightarrow \) price

\( name^+ = \{name, color\} \)

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

\( color^+ = \{color\} \)
Closure Algorithm

\[ X = \{ A_1, \ldots, A_n \} \]

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

\[ \text{if } B_1, \ldots, B_n \rightarrow C \text{ is a FD and } B_1, \ldots, B_n \text{ are all in } X \]

**then** add \( C \) to \( X \).

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

**Hence:** name, category \( \rightarrow \) color, department, price

Example:

- name \( \rightarrow \) color
- category \( \rightarrow \) department
- color, category \( \rightarrow \) price
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, \}
Why Do We Need Closure

• With closure we can find all FD’s easily

• To check if $X \rightarrow A$
  – Compute $X^+$
  – Check if $A \in X^+$
Using Closure to Infer ALL FDs

Example:

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

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

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

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

\[
\begin{align*}
AB \rightarrow CD, & \quad AD \rightarrow BC, & \quad ABC \rightarrow D, & \quad ABD \rightarrow C, & \quad ACD \rightarrow B
\end{align*}
\]
Another Example

• **Enrollment**(student, major, course, room, time)

  student $\rightarrow$ major
  major, course $\rightarrow$ room
  course $\rightarrow$ time

What else can we infer? [in class, or at home]
Keys

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

• A **key** is a minimal superkey
  – I.e. set of attributes which is a superkey and for which no subset is a superkey
Computing (Super)Keys

- Compute $X^+$ for all sets $X$
- If $X^+ = \text{all attributes}$, then $X$ is a key
- List only the minimal $X$’s
Example

Product(name, price, category, color)

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

What is the key?
Example

Product(name, price, category, color)

\[
\begin{array}{|c|c|}
\hline
\text{name, category} & \rightarrow \text{price} \\
\text{category} & \rightarrow \text{color} \\
\hline
\end{array}
\]

What is the key?

\[(\text{name, category}) + = \text{name, category, price, color}\]

Hence (name, category) is a key
Examples of Keys

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

\begin{align*}
\text{student} & \rightarrow \text{address} \\
\text{room, time} & \rightarrow \text{course} \\
\text{student, course} & \rightarrow \text{room, time}
\end{align*}

(find keys at home)
Eliminating Anomalies

Main idea:

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

• $X \rightarrow A$ is not OK otherwise
### 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>

**SSN \rightarrow Name, City**

What the key?  
\{SSN, PhoneNumber\}

Hence **SSN \rightarrow Name, City** is a “bad” dependency
Key or Keys?

Can we have more than one key?

Given $R(A,B,C)$ define FD’s s.t. there are two or more keys.
Key or Keys?

Can we have more than one key?

Given $R(A,B,C)$ define FD’s s.t. there are two or more keys

$\begin{align*}
\text{AB} &\rightarrow \text{C} \\
\text{BC} &\rightarrow \text{A}
\end{align*}$

or

$\begin{align*}
\text{A} &\rightarrow \text{BC} \\
\text{B} &\rightarrow \text{AC}
\end{align*}$

what are the keys here?

Can you design FDs such that there are three keys?
Boyce-Codd Normal Form

A simple condition for removing anomalies from relations:

A relation R is in BCNF if:

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

In other words: there are no “bad” FDs

Equivalently:

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

\textbf{repeat}
\begin{itemize}
  \item choose $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ that violates BCNF
  \item split $R$ into $R_1(A_1, \ldots, A_m, B_1, \ldots, B_n)$ and $R_2(A_1, \ldots, A_m, [\text{others}])$
  \item continue with both $R_1$ and $R_2$
\end{itemize}
\textbf{until} no more violations

\begin{itemize}
  \item Is there a 2-attribute relation that is not in BCNF?
  \item In practice, we have a better algorithm (coming up)
\end{itemize}
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>

SSN $\rightarrow$ Name, City

What the key?

\{SSN, PhoneNumber\} use SSN $\rightarrow$ Name, City to split
Example

<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 \rightarrow Name, City

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

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

SSN $\rightarrow$ name, age
age $\rightarrow$ hairColor

Decompose in BCNF (in class):
BCNF Decomposition Algorithm

BCNF_Decompose(R)

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

if (not found) then “R is in BCNF”

let $Y = X^+ - X$
let $Z = \text{[all attributes]} - X^+$
decompose R into $R_1(X \cup Y)$ and $R_2(X \cup Z)$
continue to decompose recursively $R_1$ and $R_2$
Find $X$ s.t.: $X \neq X^+ \neq [\text{all attributes}]$

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

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

**Iteration 2:** P

age+ = age, hairColor

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

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

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

**What are the keys?**
Example

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

$A^+ = ABC \neq ABCD$

$R_1(A,B,C)$

$B^+ = BC \neq ABC$

$R_{11}(B,C)$

$R_{12}(A,B)$

$R_2(A,D)$

What are the keys?

What happens if in $R$ we first pick $B^+$? Or $AB_4^+$?
Decompositions in General

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

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

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

\[ R_1 = \text{projection of } R \text{ on } A_1, ..., A_n, B_1, ..., B_m \]
\[ R_2 = \text{projection of } R \text{ on } A_1, ..., A_n, C_1, ..., C_p \]
Theory of 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>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>

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>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>

Lossy decomposition
Decompositions in General

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

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

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

If \( A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m \) Then the decomposition is lossless

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

BCNF decomposition is always lossless. WHY?