Outline

- Design theory: 3.1-3.4
  - [Old edition: 3.4-3.6]
Schema Refinements = Normal Forms

- 1st Normal Form = all tables are flat
- (2nd Normal Form = obsolete)
- Boyce Codd Normal Form = main focus
- 3rd Normal Form = see book for more details
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>Course</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>

May need to add keys

### 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

- Math
- DB
- OS

http://www.cs.washington.edu/education/courses/cse444/11wi/
Conceptual Schema Design

Conceptual Model:

Relational Model: plus FD’s (FD = Functional Dependency)

Normalization: Eliminates anomalies
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
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.

Primary key is thus (SSN, PhoneNumber)

The above is in 1NF, but what is the problem with this schema?
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>

Anomalies:
- Redundancy = repeat data
- Update anomalies = what if Fred moves to “Bellevue”?
- Deletion anomalies = what if Joe deletes his phone number? (what if Joe had only one phone #)
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 (Logical Design)

- Main idea:
  - Start with some relational schema
  - Find out its functional dependencies (discussed next!)
  - 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
- Use them to normalize the relations
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:

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

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

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

<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 $\rightarrow$ Name, Phone, Position

Position $\rightarrow$ Phone

but not: Phone $\rightarrow$ 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 $\rightarrow$ 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>

name → color
category → department
color, category → price

Does this instance satisfy all the FDs?
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>
<tr>
<td>Gizmo</td>
<td>Stationary</td>
<td>Blue</td>
<td>Supplies</td>
<td>59</td>
</tr>
</tbody>
</table>

What about this one?

http://www.cs.washington.edu/education/courses/cse444/11wi/
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

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
d
$A_1, A_2, \ldots, A_n \rightarrow B_m$

Splitting rule and Combing rule
Armstrong’s Rules (2/3)

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

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

**Trivial Rule**

Why ?

<table>
<thead>
<tr>
<th>( A_1 )</th>
<th>...</th>
<th>( A_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Transitive 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?
Armstrong’s Rules (3/3)

Illustration for Transitivity

<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>
<th>$C_1$</th>
<th>$\ldots$</th>
<th>$C_p$</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>

http://www.cs.washington.edu/education/courses/cse444/11wi/
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)

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

Answers:

<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</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</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 $A_1, ..., A_n$

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

Example: 
- name $\rightarrow$ color
- category $\rightarrow$ department
- color, category $\rightarrow$ price

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

X={A₁, ..., Aₙ}.

Repeat until X doesn’t change do:
  if B₁, ..., Bₙ → C is a FD and B₁, ..., Bₙ are all in X
  then add C to X.

Example:

name → color
category → department
color, category → price

{name, category}⁺ =
{ name, category, color, department, price }

Hence: name, category → color, department, 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 \( \{A, B\}^+ \) \( X = \{A, B, \} \)

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

In class:

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

- \[ A, B \rightarrow C \]
- \[ A, D \rightarrow E \]
- \[ B \rightarrow D \]
- \[ A, F \rightarrow B \]

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

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

In class:

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

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

Compute \( \{A, F\}^+ \) \( X = \{A, F, B, C, D, E\} \)
Why Do We Need Closure

- With closure we can find all FD’s easily

- To check if \( X \rightarrow A \)
  - Compute \( X^+ \)
  - Check if \( A \subseteq X^+ \)
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 \( X \):

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

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

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

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

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

What else can we infer? [ in class ]

Solution is on our group wiki:
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.
  - 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 superkey
- List only the minimal $X$’s to get the keys
Example

Product(name, price, category, color)

name, category → price
category → color

What is the key?
Example

Product(name, price, category, color)

name, category $\rightarrow$ price

category $\rightarrow$ color

What is the key?

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

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

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

\begin{itemize}
\item \text{student} \rightarrow \text{address}
\item \text{room, time} \rightarrow \text{course}
\item \text{student, course} \rightarrow \text{room, time}
\end{itemize}

Find keys at home!

Solution soon on our group wiki:
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>

What is 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

\[
\begin{aligned}
AB & \rightarrow C \\
BC & \rightarrow A \\
\text{or} & \\
A & \rightarrow BC \\
B & \rightarrow AC
\end{aligned}
\]

What are the keys here?
Can you design FDs such that there are *three* keys?
Boyce-Codd Normal Form (BCNF)

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

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

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

In practice, we have a better algorithm (coming up)

http://www.cs.washington.edu/education/courses/cse444/11wi/
Example (revisited)

<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 is the key? \{SSN, PhoneNumber\}

Hence \( SSN \rightarrow \text{Name, City} \) is a “bad” dependency
Example (revisited)

<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?
Example Decomposition

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

FD1: SSN $\rightarrow$ name, age
FD2: age $\rightarrow$ hairColor

*Decompose into BCNF (in class):*
Example Decomposition

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

- \( \text{FD1: SSN } \rightarrow \text{ name, age} \)
- \( \text{FD2: age } \rightarrow \text{ hairColor} \)

*Decompose into BCNF (in class):*

*What is the key?* \{SSN, phoneNumber\}

*But how to decompose?*

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

\text{or}\

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

\text{or ...}
BCNF Decomposition Algorithm

```plaintext
BCNF_Decompose(R)

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

if (not found) then “R is in BCNF”

let Y = X⁺ - X
let Z = [all attributes] - X⁺
decompose R into R₁(X ∪ Y) and R₂(X ∪ Z)
continue to decompose recursively R₁ and R₂
```
Example BCNF Decomposition

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

FD1: SSN → name, age
FD2: age → hairColor

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

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?
Example

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

$A \rightarrow B$

$B \rightarrow C$

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

$A^+ = ABC \neq ABCD$

$R_1(A,B,C)$

$B^+ = BC \neq ABC$

$R_{11}(B,C)$

$R_{12}(A,B)$

$R_2(A,D)$

What are the keys?

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

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

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

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

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

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

http://www.cs.washington.edu/education/courses/cse444/11wi/
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 ?
General Decomposition Goals

1. Elimination of anomalies

2. Recoverability of information
   - Can we get the original relation back?

3. Preservation of dependencies
   - Want to enforce FDs without performing joins

Sometimes cannot decompose into BCNF without losing ability to check some FDs in single relation
### BCNF and Dependencies

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

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>

Unit → Company

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

No FDs

In BCNF we lose the FD  
Company, Product → Unit
3NF Motivation

A relation R is in 3rd normal form if:
Whenever there is a nontrivial dep. \( A_1, A_2, ..., A_n \rightarrow B \) for R,
then \( \{A_1, A_2, ..., A_n\} \) is a super-key for R,
or B is part of a key.

Tradeoffs:

**BCNF**: no anomalies, but may lose some FDs
**3NF**: keeps all FDs, but may have some anomalies
Motivation of 4NF and higher

Assume for each course, we can independently choose a lecturer and a book. What is the problem?

<table>
<thead>
<tr>
<th>Course</th>
<th>Lecturer</th>
<th>Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>cse444</td>
<td>Alexandra</td>
<td>Complete book</td>
</tr>
<tr>
<td>cse444</td>
<td>Wolfgang</td>
<td>Complete book</td>
</tr>
<tr>
<td>cse444</td>
<td>Alexandra</td>
<td>Cow book</td>
</tr>
<tr>
<td>cse444</td>
<td>Wolfgang</td>
<td>Cow book</td>
</tr>
</tbody>
</table>

Multi-valued dependency (MVD) Course →→ Lecturer: In every legal instance, each Course value is associated with a set of Lecturer values and this set is independent of the values in the other attributes (here Book).