Schema Anomalies

• Redundant storage
  – Repeatedly storing the same information

• Update anomaly
  – To change a repeated item, every occurrence should be changed

• Insertion anomaly
  – Some information cannot be stored without additional (possibly unavailable) information

• Deletion anomaly
  – Some information cannot be deleted without deleting additional (possibly desired) information

From ERD to Normalization

• We have learned how to design schemas using ERDs
• But it is often not enough for a proper translation into well designed relations
• ERD is limited in constraint representation; we need a more careful design to enforce such constraints
• It may be challenging to avoid anomalies when dependencies are complicated
Example

- A track has at most one consultant per faculty
- A track is contained in a single campus
- A consultant belongs to a single campus and faculty
- A faculty is in a single campus

The Refined Design Process (Normalization)

1. Define the involved attributes
2. Determine what constraints / dependencies hold in real life
3. Decide on desired properties
4. Decompose into multiple good ("normalized") schemas
Notation

- During this lecture we view a relation schema as a pair \((U, F)\) where:
  - \(U\) is a finite set of attributes
  - \(F\) is a set of FDs over \(U\)
- In particular, we ignore:
  - relation names
  - order among attributes

Basic Terminology

- Let \((U, F)\) be a relation schema
- Recall: A superkey is a set \(K\) of attributes such that \(K^+\) contains every attribute in \(U\)
- Recall: A key is a superkey \(K\) that does not contain any other superkey
  - That is, if \(Y \subseteq K\) then \(Y\) is not a superkey
- Attributes of keys are called prime
- “Schema normalization” constrains the relationship between FDs, keys, prime attributes and nonprime attributes

History of Normal Forms

1NF 2NF 3NF BCNF 4NF 5NF

Nonprime attributes are not dependent on strict parts of keys

“Standard” normal form: a nonprime attribute can be determined only by a superkey

A relation does not involve any “implicit” joins

No nontrivial MVDs except for superkey

No nontrivial FDs except for superkeys

DB looks like a logical structure; assumed by default
Our Focus

- We mainly focus on BCNF and 3NF
  - Historically BCNF came after 3NF, but we start with BCNF since it is simpler
- In the end we will briefly review 4NF
Boyce-Codd Normal Form (BCNF)

- A schema \((U,F)\) is in BCNF if every nontrivial FD implied by \(F\) has a superkey on its premise (lhs)
- That is, every \(X \rightarrow Y\) in \(F^+\) is such that
  - \(X\) is a superkey; or
  - \(Y \subseteq X\)

Examples

Faculty: name, symbol, dean

- name \rightarrow symbol, symbol \rightarrow dean, dean \rightarrow name

Social network: follows, followed, fid

- follows, followed \rightarrow fid, fid \rightarrow follows, followed

Address: state, city, street, zip

- state, city, street \rightarrow zip, zip \rightarrow state

Tracks: track, faculty, consultant, campus

- track, faculty, consultant, campus, consultant \rightarrow faculty,
  - track \rightarrow campus, faculty \rightarrow campus

Can BCNF be Tested Efficiently?

- On the face of it, we need to consider every derived FD (exponentially many); however:
- **Theorem**: The following are equivalent:
  1. The schema \((U,F)\) is in BCNF (i.e., every nontrivial \(X \rightarrow Y\) in \(F^+\) is such that \(X\) is a superkey)
  2. In every nontrivial \(X \rightarrow Y\) in \(F\), \(X\) is a superkey
- Hence, it suffices to check \(F\)
- Proof not given
  - *But which direction is straightforward?*
- *So what would be an efficient BCNF testing?*
Outline

- Introduction
- Normal Forms
  - BCNF
  - 3NF
- Decomposition
  - NF Decompositions
  - Preserving Data
  - Preserving Dependencies
- Decomposition Algorithms
  - 3NF
  - BCNF
  - Note on 4NF

---

Third Normal Form (3NF)

- Recall: an attribute $A$ is prime if it is a part of some key
  - Warning: not “superkey;” every attribute belongs to some superkey
- A schema is in 3NF if for every nonprime $A$ and nontrivial derived $X \rightarrow A$, the set $X$ is a superkey
- Equivalently, for every $X \rightarrow A$ in $F^+$ at least one of the following holds:
  - $X$ is a superkey
  - $A \in X$
  - $A$ is prime

---

Examples

Faculty:

<table>
<thead>
<tr>
<th>name, symbol, dean</th>
<th>BCNF 3NF</th>
</tr>
</thead>
<tbody>
<tr>
<td>name—symbol, symbol—dean, dean—name</td>
<td>3NF</td>
</tr>
</tbody>
</table>

Social network:

<table>
<thead>
<tr>
<th>follows,followed,fid</th>
<th>BCNF 3NF</th>
</tr>
</thead>
<tbody>
<tr>
<td>follows,followed—fid, fid—followed</td>
<td>3NF</td>
</tr>
</tbody>
</table>

Address:

<table>
<thead>
<tr>
<th>state,city,street,zip</th>
<th>not BCNF 3NF</th>
</tr>
</thead>
<tbody>
<tr>
<td>state,city,street—zip, zip—state</td>
<td>3NF</td>
</tr>
</tbody>
</table>

Tracks:

<table>
<thead>
<tr>
<th>track,faculty,consultant,campus</th>
<th>not BCNF not 3NF</th>
</tr>
</thead>
<tbody>
<tr>
<td>track,faculty—consultant, consultant—faculty, faculty—campus</td>
<td>not BCNF</td>
</tr>
</tbody>
</table>
Testing 3NF?

- The following algorithm works:
- For every nontrivial FD $X \rightarrow Y$ in $F$
  1. Check whether $X$ is a superkey
  2. Check whether every attribute in $Y \setminus X$ is prime
- As we know, (1) can be tested efficiently
- What about (2)?
  - It is NP-complete! (unlikely to be solvable in polynomial time)
- And in fact, testing whether a schema is in 3NF is an NP-complete problem [Jou, Fischer 82]

Outline

- Introduction
- Normal Forms
  - BCNF
  - 3NF
- Decomposition
  - NF Decompositions
  - Preserving Data
  - Preserving Dependencies
- Decomposition Algorithms
  - 3NF
  - BCNF
  - Note on 4NF

Decomposition

- We can fix a “badly designed” schema by decomposing it into several smaller schemas
- But we need to do so correctly!
  - Do not change our intended information
  - Do not violate the FDs
  - Get a “well designed” fixed schema
- In this part, we will make the above formal
- First, we need a notation
Restricting a Set of FDs

- Let \((U, F)\) be a schema
- Let \(W\) be a subset of \(U\)
- We denote by \(F[W]\) the set of all the FDs \(X \rightarrow Y\) in \(F\) such that \(XY \subseteq W\)

Formal Definition

- A decomposition of a schema \((U, F)\) is a collection \((X_1, F_1), \ldots, (X_k, F_k)\) of schemas such that:
  - \(U = X_1 \cup \cdots \cup X_k\)
  - That is, the \(X_i\) cover all the attributes in \(U\)
  - For \(i = 1, \ldots, k\) we have \((F_i) = F[X_i]\)
  - That is, each \(F_i\) consists of the FDs imposed by \(F\) on \(X_i\)

Decomposing and Composing Relations

\[(U, F) \xrightarrow{\sigma} (X_1, F_1) \xrightarrow{\sigma} (X_2, F_2) \xrightarrow{\sigma} \cdots \xrightarrow{\sigma} (X_k, F_k)\]
Representing $F_i$

- Given the schema $(U,F)$, it suffices to represent a decomposition using the collection $\{X_1, \ldots, X_k\}$ without mentioning the FDs $F_i$.
- Since $F_i$ can be $f[X]$ up to equivalence.
- Problem: naively constructing $F_i$ as $F^+[X]$ can be expensive, since $F^+$ and $F^+[X]$ can be exponentially larger than $U$.
  - This problem is unavoidable: It may be that $F^+[X]$ is not equivalent to any sub-exponential #FDs!
  - We keep this problem in mind – we will not assume that $F^+[X]$ can be materialized efficiently.

Outline

- Introduction
- Normal Forms
  - BCNF
  - 3NF
- Decomposition
  - NF Decompositions
  - Preserving Data
  - Preserving Dependencies
- Decomposition Algorithms
  - 3NF
  - BCNF
  - Note on 4NF

Obtaining Normal Forms

- Let $N$ be a normal form (e.g., 3NF, BCNF).
- An $N$ decomposition of a schema $(U,F)$ is a decomposition $(X_1, \ldots, X_k)$ of $(U,F)$ such that each $(X_i, F[X_i])$ is in $N$.
- We will discuss 3NF decompositions and BCNF decompositions.
Examples

3NF decomposition? BCNF decomposition?

\[ \text{ABCD} \xrightarrow[A→B, B→C, \quad \text{AD}→D, \text{D}→B}{A→D, \text{BC}→\text{C}, \quad \text{BD}→\text{B}} \]
Answer: BCNF, 3NF

\[ \text{ABCD} \xrightarrow[\text{AB}→\text{C}, \quad \text{C}→\text{B}]{\text{AB}→\text{C}, \quad \text{C}→\text{B}} \]
Answer: 3NF, not BCNF

\[ \text{ABCD} \xrightarrow[A→B, \quad \text{B}→\text{C}, \quad \text{C}→\text{D}, \quad \text{A}→\text{C}]{A→\text{D}, \quad \text{ACD}→\text{D}} \]
Answer: not 3NF, not BCNF

Outline

• Introduction
• Normal Forms
  • BCNF
  • 3NF
• Decomposition
  • NF Decompositions
  • Preserving Data
  • Preserving Dependencies
• Decomposition Algorithms
  • 3NF
  • BCNF
  • Note on 4NF

Good Decomposition?

Can you restore?

person → building/room

<table>
<thead>
<tr>
<th>person</th>
<th>building</th>
<th>room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>Taub</td>
<td>152</td>
</tr>
<tr>
<td>Amir</td>
<td>Meyer</td>
<td>35</td>
</tr>
<tr>
<td>Ahuva</td>
<td>Meyer</td>
<td>246</td>
</tr>
</tbody>
</table>
Lossless Decomposition

- Let \( \{X_1, \ldots, X_k\} \) be a decomposition of \((U, F)\).
- We say that \( \{X_1, \ldots, X_k\} \) is a lossless decomposition of \((U, F)\) if for all relations \( r \) over \((U, F)\) we have:
  \[ \pi_{X_1}(r) \Join \cdots \Join \pi_{X_k}(r) = r \]
- Containment in one direction always holds:
  \[ \pi_{X_1}(r) \Join \cdots \Join \pi_{X_k}(r) \supseteq r \]
- What about the other direction? Depends on \( F \)!

Example 1

Example 2
Decision Algorithm

**Losslessness Testing**

**Given:**
- \( U, F, X_1, \ldots, X_k \)
- \( (X_1, \ldots, X_k) \) is a decomposition of \( (U,F) \)

**Goal:**
- Determine whether \( (X_1, \ldots, X_k) \) is a lossless decomposition

- The definition of *lossless* is not constructive (check every possible relation)
- Next, we present a polynomial-time algorithm for this decision problem

---

**The Case of Binary Decomposition**

**THEOREM:** Let \( \{X_1, X_2\} \) be a decomposition of \( (U,F) \). The following are equivalent:
1. \( F \models (X_1 \cap X_2) \rightarrow X_1 \) or \( F \models (X_1 \cap X_2) \rightarrow X_2 \)
2. \( (X_1, X_2) \) is a lossless decomposition

*So what would be a decision algorithm in this case?*

---

**Proof: 1\( \Rightarrow \)2**

1. \( F \models (X_1 \cap X_2) \rightarrow X_1 \) or \( F \models (X_1 \cap X_2) \rightarrow X_2 \)
2. \( (X_1, X_2) \) is a lossless decomposition

We need to prove that \( 1 \) is here!

We know that this is a subset of \( r \), for some \( x \)

In any case, we had \( 1 \) to begin with, QED

Hence lossless!
Proof: not 1 $\Rightarrow$ not 2

1. $F = X_1 \cap X_2 \rightarrow X_1$ or $F = X_1 \cap X_2 \rightarrow X_2$
2. $(X_1, X_2)$ is a lossless decomposition

- Let $X_{12} = (X_1 \cap X_2)^*$ and suppose that $X_1 \not\subseteq X_{12}$ and $X_2 \not\subseteq X_{12}$
- Construct a relation $r = \{t, u\}$ over $U$:
  - $t[X_{12}] = \{0, \ldots, 0\}$
  - $t[U \setminus X_{12}] = \{1, \ldots, 1\}$
  - $u[X_{12}] = \{2, \ldots, 2\}$
  - $u[U \setminus X_{12}] = \{0, \ldots, 0\}$
- Claim 1: $r \models F$
  - Proof similar to completeness of Armstrong’s axioms
- Claim 2: $\pi_{X_1}(r) \Join \pi_{X_2}(r) \neq r$
  - The join contains a row with both 1s and 2s

Illustration: not 1 $\Rightarrow$ not 2

1. $F = X_1 \cap X_2 \rightarrow X_1$ or $F = X_1 \cap X_2 \rightarrow X_2$
2. $(X_1, X_2)$ is a lossless decomposition

The General Case

Losslessness Testing

<table>
<thead>
<tr>
<th>Given:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U$, $F$, $X_1 \rightarrow X_5$</td>
</tr>
<tr>
<td>$[X_1 \rightarrow X_5]$ is a decomposition of $(U,F)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Goal:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determine whether $(X_1 \rightarrow X_5)$ is a lossless decomposition</td>
</tr>
</tbody>
</table>

Next, we handle the general case of a decomposition ($\geq 2$ schemas)
The General Case

### Losslessness Testing

<table>
<thead>
<tr>
<th>Given:</th>
<th>Goal:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• (U, F, X \rightarrow Y)</td>
<td>Determine whether ((X \rightarrow Y)) is a lossless decomposition</td>
</tr>
</tbody>
</table>

1. **1st step: create the "known subset"**
   - A table over \(U\), one tuple \(t_i\) for each \(X \rightarrow Y\), if \(X\) contains \(A_j\) and \(t_i[A_j] = a_i\) otherwise

2. **2nd step: chase**
   - While the table changes do:
     - Look for an FD violation and equate the conclusions
     - "Equate" = change every occurrence of one to the other
     - When equating \(a_i\) with \(x_j\), change \(x_j\) to \(a_i\)

3. **3rd step: Return true iff there is a row of \(a_i\)'s**

---

### Negative Example

- **We need to prove:**

- **Lossy!** (how do we know that?)

---

### The Idea

- **We need to prove:**

- **But some of the x's may be known due to the FDs!**

---

### The General Case

- **We know that this is a subset of \(x\), for some x's**

---

---
**Example**

\[ T = \{ A_2 \rightarrow A_4, A_4 \rightarrow A_5, A_5 \rightarrow A_2 \} \]

**Step 1:** construct the known subset

**Step 2:** chase

**Step 3:** return true

---

**Think**

*How do we generalize the proof of correctness from the two-table case?*

*Why is this algorithm terminating in polynomial time?*

---

**Outline**

- Introduction
- Normal Forms
  - BCNF
  - 3NF
- Decomposition
  - NF Decompositions
  - Preserving Data
  - Preserving Dependencies
- Decomposition Algorithms
  - 3NF
  - BCNF
  - Note on 4NF
### Preserving Dependencies

Is $F$ preserved given that each $F_i$ is preserved in each relation?

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(U,F)</td>
<td>(X,F)</td>
<td>(X,F)</td>
</tr>
<tr>
<td></td>
<td>r_1</td>
<td>r_2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Example 1

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A→B, B→C, ABC→D, D→B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AD</td>
<td>BC</td>
<td>BD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A→D</td>
<td>B→C</td>
<td>D→B</td>
<td></td>
</tr>
</tbody>
</table>

Are dependencies preserved in this decomposition?  
Answer: Yes!

### Example 2

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A→B</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BC</td>
<td>AC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C→B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Are dependencies preserved in this decomposition?  
Answer: No! 

Is there any decomposition into binary schemas where dependencies are preserved?  
Answer: No!
**Formal Definition**

- A decomposition $X_1, ..., X_k$ of $(U,F)$ is dependency preserving if for all $r_1, ..., r_k$ over $(X_1,F_1), ..., (X_k,F_k)$, respectively, where $F_i = F[X_i]$, the relation $r_1 \bowtie \cdots \bowtie r_k$ satisfies $F$.

- Can we test whether a given decomposition has this property?

**THEOREM:** The following are equivalent:

1. For all $r_1, ..., r_k$ over $(X_1,F_1), ..., (X_k,F_k)$, respectively, the relation $r_1 \bowtie \cdots \bowtie r_k$ satisfies $F$.
2. $F^+ = (F_1 \cup \cdots \cup F_k)^+$

**Testing for Dependency Preservation**

- We need to test whether $F^+ = (F_1 \cup \cdots \cup F_k)^+$.
- $F^+ \supseteq F_1 \cup \cdots \cup F_k$, so $F^+ \supseteq (F_1 \cup \cdots \cup F_k)^+$.
- So, need to test whether $F^+ \subseteq (F_1 \cup \cdots \cup F_k)^+$.
- It suffices to test whether each $X \rightarrow Y$ in $F$ is implied by $F_1 \cup \cdots \cup F_k$.
  - Or in other words, whether $Y$ is a subset of the closure of $X$ under $F_1 \cup \cdots \cup F_k$.
- Next slide: efficient computation of the closure of $X$ under $F_1 \cup \cdots \cup F_k$.
  - Without explicitly calculating the $F_i$'s!

**Closure w.r.t. a Decomposition**

**Given:**
- $U, F, X_1, ..., X_k, X$
- $(X_1, ..., X_k)$ is a decomposition of $(U,F)$
- $X \subseteq U$

**Goal:** Compute the closure of $X$ under $F^+[X \cup -U\cap F[X]]$.

**ClosureDecomp**($X,F,X_1,\ldots,X_k$) {
  $Y := X$
  while($Y$ changes)
    for($i = 1, \ldots, k$)
      $Y := Y \cup (\text{Closure}(Y \cap X_i) \cap X_i)$
  return $Y$
}
### Testing for Dependency Preservation

**Dependency Preservation Testing**

<table>
<thead>
<tr>
<th>Given:</th>
<th>Goal:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( U, F, X_1, \ldots, X_k )</td>
<td>Determine whether ( {X_1, \ldots, X_k} ) is dependency preserving</td>
</tr>
<tr>
<td>( {X_1, \ldots, X_k} ) is a decomposition of ( U,F )</td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
DepPreserving(X_1, \ldots, X_k, F) {
    for all \( X \rightarrow Y \) in F
    if \( Y \not\subseteq \text{ClosureDecomp}(X, F, X_1, \ldots, X_k) \)
    return false
    return true
}
```

---

### Outline

- Introduction
- Normal Forms
  - BCNF
  - 3NF
- Decomposition
  - NF Decompositions
  - Preserving Data
  - Preserving Dependencies
- Decomposition Algorithms
  - 3NF
  - BCNF
  - Note on 4NF

---

### Decomposition Algorithms

- Given a normal form \( N \), we ask:
  - Is there always a lossless \( N \) decomposition?
  - Is there always a lossless & dependency preserving \( N \) decomposition?
  - Is there an efficient decomposition?
- The next slides discuss two algorithms
  - 3NF decomposition
    - Lossless, dependency preserving, p-time
  - BCNF decomposition
    - Lossless
      - Not covered in the official course material
Outline

- Introduction
- Normal Forms
  - BCNF
  - 3NF
- Decomposition
  - NF Decompositions
  - Preserving Data
  - Preserving Dependencies
- Decomposition Algorithms
  - 3NF
  - BCNF
  - Note on 4NF

Intuition

Idea: for dependency preservation, each $X \rightarrow A$ becomes a schema

$$F = \{A \rightarrow B, AB \rightarrow C, C \rightarrow B, D \rightarrow C\}$$

<table>
<thead>
<tr>
<th>AB</th>
<th>ABC</th>
<th>BC</th>
<th>CD</th>
<th>AD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Problem: not in 3NF
Solution: minimal cover instead of $F$

Problem: lossy
Solution: add a key

Reminder: Minimal Cover

- Let $F$ be a set of FDs
- A **minimal cover** of $F$ is a set $G$ of FDs such that $G^+ = F^+$ with the following properties:
  - FDs in $G$ have a single attribute on the right hand side; that is, they have the form $X \rightarrow A$
  - All FDs are required: no FD $X \rightarrow A$ in $G$ is such that $G \setminus \{X \rightarrow A\} \vDash X \rightarrow A$
  - All attributes are required: no FD $XB \rightarrow A$ in $G$ is such that $G \setminus X \rightarrow A$
- Exercise: Suggest an algorithm for computing a minimal cover
Revised Example

\{ A \rightarrow B, AB \rightarrow C, C \rightarrow B, D \rightarrow C \} \\

\[ F = \{ A \rightarrow C, C \rightarrow B, D \rightarrow C \} \]

Algorithm for 3NF Decomposition

\[
3\text{NFD}ec(U, F) \{
\begin{align*}
D &= \emptyset \\
G &:= \text{MinCover}(F) \\
\text{for all } & (X \rightarrow A \text{ in } G) \text{ do} \\
D &:= D \cup \{XA\} \\
\text{if (no set in } D \text{ is a superkey)} \text{ then} \\
D &= D \cup \{\text{FindKey}(U, F)\} \\
D &:= \text{RemoveContained}(D) \\
\text{return } D
\end{align*}
\]

About the Proof

- We will not prove the correctness here
- Still, what needs to be proved?
  - Resulting schemas are all in 3NF
    - Follows from minimality of the cover
  - Dependencies are preserved
    - Straightforward: all dependencies of the minimal cover are presented
  - Lossless
    - What would the lossless-testing algorithm do when one \( X_i \) is a key and all dependencies preserved?
Example Revisited

The rest of the presentation is not in the official course material.

OPTIONAL MATERIAL

Outline

- Introduction
- Normal Forms
  - BCNF
  - 3NF
- Decomposition
  - NF Decompositions
  - Preserving Data
  - Preserving Dependencies
- Decomposition Algorithms
  - 3NF
  - BCNF
  - Note on 4NF
Key Insight

- Recall: BCNF means that in every nontrivial
  \( X \rightarrow Y \), the set \( X \) is a superkey
- CLAIM: If \((U,F)\) is not in BCNF, then there is a
  lossless decomposition \( \{X_1, X_2\} \) with \( X_1, X_2 \subseteq U \)
- Proof:
  - Let \( X \rightarrow Y \) be a BCNF violation (\( X \) is not a superkey
    and \( Y \) is not a subset of \( X \))
  - Take \( X = X' \) and \( X \rightarrow X' \cup (U \setminus X') \)
  - Why are \( X_1 \) and \( X_2 \) strict subsets of \( U \)?
  - Why lossless?
    - Recall the theorem on binary lossless decompositions ...

BCNF Decomposition

\[
\begin{align*}
\text{BCNFD} & (U,F) \{ \\
\text{if } (U,F) \text{ in BCNF} & \quad \text{return } (U) \\
& \quad \text{Find a BCNF violation } X \rightarrow Y \\
& \quad X_1 := \text{Closure}(X,F) \\
& \quad F_1 := F' [X_1] \\
& \quad X_2 := X \cup U \setminus X_1 \\
& \quad F_2 := F' [X_2] \\
& \quad \text{return BCNFD}(X_1,F_1) \cup \text{BCNFD}(X_2,F_2)
\end{align*}
\]

Execution Example

Are dependencies preserved in this decomposition?

\[\text{Answer: Yes, we already saw that previously}\]
**About the Algorithm**

- **Lossless**
  - Proof idea: every step is lossless
- **Exponential time** in the worst case
- **There is a polynomial-time algorithm for BCNF decomposition**
  - [Tsou, Fischer, Decomposition of a relation scheme into Boyce-Codd Normal Form, 1982]
- **The algorithm does not preserve dependencies!**
  - But the problem is not with the algorithm...

**Can Dependencies be Preserved?**

\[
\begin{align*}
\text{ABC} &\quad \text{AB} \rightarrow C \quad C \rightarrow B \\
\text{BC} &\quad \text{C} \rightarrow B \\
\text{AC} &
\end{align*}
\]

No BCNF decomposition of this schema preserves both dependencies (why?)

Conclusion: Lossless BCNF decomposition is always possible. Lossless & dependency-preserving BCNF decomposition may be impossible.

**Outline**

- Introduction
- Normal Forms
  - BCNF
  - 3NF
- Decomposition
  - NF Decompositions
  - Preserving Data
  - Preserving Dependencies
- Decomposition Algorithms
  - 3NF
  - BCNF
  - Note on 4NF
Fourth Normal Form (4NF)

- **Recall**: An MVD has the form $X \rightarrow Y$ where $X$ and $Y$ are disjoint sets of attributes.
  - For every two tuples that agree on $X$, swapping their $Y$ component doesn’t change the relation.
- **Recall**: An MVD $X \rightarrow Y$ is trivial (always holds) if and only if $Y = \emptyset$ or $Y = U \setminus X$.
- **Recall**: an FD $X \rightarrow Y$ can be viewed as a special type of the MVD $X \rightarrow Y$ (why?)
- A schema $(U, F)$, where $F$ contains both FDs and MVDs, is in 4NF if every nontrivial FD/MVD has a superkey in its premise (LHS).
  - When all dependencies are FDs, same as BCNF.

4NF Decomposition

- **THEOREM**: Let $(U, F)$ be a schema, where $F$ contains both FDs and MVDs. Then $F$ satisfies $X \rightarrow Y$ iff for all relations $r$ over $U$ we have:
  $$ r = \pi_{X \cup Y}(r) \bowtie \pi_{X \cup (U \setminus Y)}(r) $$
- Hence, the recursive decomposition algorithm for BCNF decomposition works here:
  - Decompose $(X \cup Y) \cup$ Decompose $(X \cup (U \setminus Y))$
  - A polynomial time is known for special cases.
- In particular, there is always a lossless 4NF decomposition.
  - **Answer**: No! Even if there are only FDs (recall BCNF).