Lecture 7:
Schema Normalization
• 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
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.
- 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

What makes it “good”?
Are there principles to follow?
Can design be automated?
The Refined Design Process (Normalization)

1. Define the involved attributes

2. Determine what dependencies hold in real life

2a. Determine implied dependencies

Which FDs allowed? Should all dependencies be enforced?

3. Decide on desired properties

4. Decompose into multiple good (“normalized”) schemas
During this lecture, we focus on schemas of a special type: a single relation over a set $U$ of attributes, and a set $F$ of FDs.

So, during this lecture a schema is simply a pair $(U,F)$ where:

- $U$ is a finite set of attributes
- $F$ is a set of FDs over $U$

(In particular, we ignore the relation name and order among attributes)
Basic Terminology

• Let \((U,F)\) be a schema
• Recall: A \textit{superkey} is a set \(K\) of attributes such that \(K^+\) contains every attribute in \(U\)
• Recall: A \textit{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 \textit{prime}
• “Schema normalization” deals with the relationship between keys, prime attributes and nonprime attributes
History of Normal Forms

1NF: 1970
- Nonprime attributes are not dependent on strict parts of keys
- DB looks like a logical structure; assumed by default

2NF: 1971
- "Standard" normal form: a nonprime attribute can be determined only by a superkey

3NF: 1974
- A relation does not involve any "implicit" joins
- "Standard" normal form: a nonprime attribute can be determined only by a superkey

BCNF: 1974
- No nontrivial MVDs except for superkey

4NF: 1977
- No nontrivial FDs except for superkeys

5NF: 1979
- A relation does not involve any "implicit" joins

1970
1971
1974
1977
1979

Codd
Boyce & Codd
Fagin
Outline

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

• Introduction
• Normal Forms
  - BCNF
  - 3NF
• Decomposition
  - NF Decompositions
  - Preserving Data
  - Preserving Dependencies
• Decomposition Algorithms
  - BCNF
  - 3NF
  - Note on 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 → symbol, symbol → dean, dean → name

Social network:

follows, followed, fid

follows, followed → fid, fid → follows, followed

Address:

state, city, street, zip

state, city, street → zip, zip → state

Tracks:

track, faculty, consultant, campus

track, faculty → consultant, consultant → faculty, track → campus, faculty → 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?**
  
  *Answer: For each \(X \rightarrow Y\) in \(F\), test whether \(U=\text{Closure}(F,X)\)*
• Introduction

• Normal Forms
  ▪ BCNF
  ▪ 3NF

• Decomposition
  ▪ NF Decompositions
  ▪ Preserving Data
  ▪ Preserving Dependencies

• Decomposition Algorithms
  ▪ BCNF
  ▪ 3NF
  ▪ Note on 4NF
Third Normal Form (3NF)

- Recall: an attribute $A$ is *prime* if it is a part of some key
- 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: name, symbol, dean
name → symbol, symbol → dean, dean → name

Social network: follows, followed, fid
follows, followed → fid, fid → follows, followed

Address: state, city, street, zip
state, city, street → zip, zip → state

Tracks: track, faculty, consultant, campus
track, faculty → consultant, consultant → faculty,
track → campus, faculty → campus
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 \textit{NP-complete}! (hence, it is unlikely that it is solvable in polynomial time)
  • And in fact, testing whether a schema is in 3NF is an NP-complete problem [JouFischer1982]
Outline

• Introduction

• Normal Forms
  ▪ BCNF
  ▪ 3NF

• Decomposition
  ▪ NF Decompositions
  ▪ Preserving Data
  ▪ Preserving Dependencies

• Decomposition Algorithms
  ▪ BCNF
  ▪ 3NF
  ▪ 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, and 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

\[(U,F) \quad (X_1,F_1) \quad (X_2,F_2) \quad (X_3,F_3) \quad \ldots \quad (X_k,F_k)\]

\[r \quad r_1 \quad r_2 \quad r_3 \quad \ldots \quad r_k\]
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 \( F_i \) is \( F[X_i] \) up to equivalence
- Problem: naively constructing \( F_i \) as \( F^+[X_i] \) can be impractical, since \( F^+ \) and \( F^+[X_i] \) can be exponentially larger than \( U \)
- But this problem is solvable! We can efficiently construct \( F_i \)'s that satisfy \((F_i)^+ = F^+[X_i]\)
  - Next slide
RestrictFDs($X_i,F$) {

$F_i := \emptyset$

for all $(Y\rightarrow Z$ in $F)$

if ($Y \subseteq X_i$)

$W := \text{Closure}(Y,F) \cap X_i$

$F_i := F_i \cup \{Y\rightarrow W\}$

return $F_i$

}

(We do not prove the correctness of the algorithm)
Outline

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

• Let $\mathbf{N}$ be a normal form (e.g., 3NF, BCNF)
• An $\mathbf{N}$ 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 $\mathbf{N}$
• We will discuss 3NF decompositions and BCNF decompositions
3NF decomposition? BCNF decomposition?

Answer: BCNF, 3NF

Answer: 3NF, not BCNF

Answer: not 3NF, not BCNF
• Introduction

• Normal Forms
  - BCNF
  - 3NF

• Decomposition
  - NF Decompositions
  - Preserving Data
  - Preserving Dependencies

• Decomposition Algorithms
  - BCNF
  - 3NF
  - Note on 4NF
Good Decomposition?

<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>

person ➔ building, room

<table>
<thead>
<tr>
<th>person</th>
<th>building</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>Taub</td>
</tr>
<tr>
<td>Amir</td>
<td>Meyer</td>
</tr>
<tr>
<td>Ahuva</td>
<td>Meyer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>building</th>
<th>room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taub</td>
<td>152</td>
</tr>
<tr>
<td>Meyer</td>
<td>35</td>
</tr>
<tr>
<td>Meyer</td>
<td>246</td>
</tr>
</tbody>
</table>
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

<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>

person→building,room

=?

=?
Example 2

<table>
<thead>
<tr>
<th>person</th>
<th>building</th>
<th>room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>Taub</td>
<td>t152</td>
</tr>
<tr>
<td>Amir</td>
<td>Meyer</td>
<td>m35</td>
</tr>
<tr>
<td>Ahuva</td>
<td>Meyer</td>
<td>m246</td>
</tr>
</tbody>
</table>

Person → building, room

Room → building

building → room

Person → room

Building → room
Decision Algorithm

Losslessness Testing

<table>
<thead>
<tr>
<th>Given:</th>
<th>Goal:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• U, F, X₁,...,Xₖ</td>
<td>Determine whether</td>
</tr>
<tr>
<td>• {X₁,...,Xₖ} is a decomposition</td>
<td>{X₁,...,Xₖ} is a lossless</td>
</tr>
<tr>
<td>of (U,F)</td>
<td>decomposition</td>
</tr>
</tbody>
</table>

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

Answer: test whether Closure\((F, X_1 \cap X_2)\) contains either \(X_1\) or \(X_2\)
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 know that this is a subset of $r$, for some $x$'s.

In any case, we had $t$ to begin with... hence lossless.
Proof: not 1 \Rightarrow not 2

1. \( F \not\vdash X_1 \cap X_2 \rightarrow X_1 \) or \( F \not\vdash X_1 \cap X_2 \rightarrow X_2 \)
2. \( \{X_1, X_2\} \) is a lossless decomposition

- Let \( Y = (X_1 \cap X_2)^+ \) and suppose that \( X_1 \not\subseteq Y, X_2 \not\subseteq Y \)
- Construct a relation \( r = \{t, u\} \) over \( U \):
  - \( t[Y] = u[Y] = (0, \ldots, 0) \)
  - \( t[U \setminus Y] = (1, \ldots, 1) \); \( u[U \setminus Y] = (2, \ldots, 2) \)

- 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. $F \models X_1 \cap X_2 \rightarrow X_1$ or $F \models X_1 \cap X_2 \rightarrow X_2$
2. $\{X_1, X_2\}$ is a lossless decomposition

\[
\begin{array}{ccccc}
A_1 & A_2 & A_3 & A_4 & A_5 \\
1 & 0 & 0 & 0 & 1 \\
2 & 0 & 0 & 0 & 2 \\
\end{array}
\rightarrow
\begin{array}{cc}
A_1 & A_2 \\
1 & 0 \\
2 & 0 \\
\end{array}
\]

$A_2 \rightarrow A_4$, $A_2 A_5 \rightarrow A_1$
Next, we handle the general case of a decomposition \((\geq 2 \text{ schemas})\).
The Idea

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

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

But some of the \( x \)'s may be known due to the FDs!

\[ F = \{ A_3 \rightarrow A_5, A_4 \rightarrow A_5, A_5 \rightarrow A_2A_4 \} \]
The General Case

Losslessness Testing

<table>
<thead>
<tr>
<th>Given:</th>
<th>Goal:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• U, F, X₁,...,Xₖ</td>
<td>Determine whether</td>
</tr>
<tr>
<td>• {X₁,...,Xₖ} is a decomposition of (U,F)</td>
<td>{X₁,...,Xₖ} is a lossless</td>
</tr>
<tr>
<td></td>
<td>decomposition</td>
</tr>
</tbody>
</table>

• **1<sup>st</sup> step: create the “known subset”**
  - A table over U, one tuple tᵢ for each Xᵢ: tᵢ[Aⱼ]=aⱼ if Xᵢ contains Aⱼ, and tᵢ[Aⱼ]=xᵢⱼ otherwise

• **2<sup>nd</sup> 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 aⱼ with xᵢⱼ, change xᵢⱼ to aⱼ

• **3<sup>rd</sup> step: Return true iff there is a row of aᵢ’s**
Example

\[ F = \{ A_3 \rightarrow A_5, \ A_4 \rightarrow A_5, \ A_5 \rightarrow A_2 A_4 \} \]

<table>
<thead>
<tr>
<th>( A_1 )</th>
<th>( A_2 )</th>
<th>( A_3 )</th>
<th>( A_4 )</th>
<th>( A_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>( x_{12} )</td>
<td>( a_3 )</td>
<td>( x_{14} )</td>
<td>( x_{15} )</td>
</tr>
<tr>
<td>( x_{21} )</td>
<td>( a_2 )</td>
<td>( a_3 )</td>
<td>( a_4 )</td>
<td>( x_{25} )</td>
</tr>
<tr>
<td>( x_{31} )</td>
<td>( x_{32} )</td>
<td>( x_{33} )</td>
<td>( a_4 )</td>
<td>( a_5 )</td>
</tr>
</tbody>
</table>

**Step 1:** construct the known subset

**Step 2:** chase

**Step 3:** return \text{true}
Why is this algorithm terminating in polynomial time?

Answer: Each iteration eliminates one symbol, and we have a polynomial #symbols
Outline

• Introduction

• Normal Forms
  ▪ BCNF
  ▪ 3NF

• Decomposition
  ▪ NF Decompositions
  ▪ Preserving Data
  ▪ Preserving Dependencies

• Decomposition Algorithms
  ▪ BCNF
  ▪ 3NF
  ▪ Note on 4NF
Preserving Dependencies

Is $F$ preserved given that each $F_i$ is preserved in each relation?
Example 1

**ABCD**

A → B, B → C, ABC → D, D → B

**{AD, BD, BC}**

**AD**

A → D

**BC**

B → C

**BD**

D → B

Are dependencies preserved in this decomposition?

**Answer:** Yes!
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 \textit{dependency preserving} if for all relations $r_1, ..., r_k$ over $(X_1,F[X_1]), ..., (X_k,F_k[X_k])$, respectively, $r_1 \Join \cdots \Join r_k$ satisfies $F$.

- \textit{Can we test whether a given decomposition has this property?}

- **THEOREM:** The following are equivalent:
  1. For all $r_1, ..., r_k$ over $(X_1,F[X_1]),..., (X_k,F_k[X_k])$, respectively, the join $r_1 \Join \cdots \Join r_k$ satisfies $F$.
  2. $F^+ = (F_1 \cup \cdots \cup F_k)^+$.
Testing for Dependency Preservation

Dependency Preservation Testing

<table>
<thead>
<tr>
<th>Given:</th>
<th>Goal:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• U, F, X₁,...,Xₖ</td>
<td>Determine whether {X₁,...,Xₖ} is dependency</td>
</tr>
<tr>
<td>• {X₁,...,Xₖ} is a decomposition of (U,F)</td>
<td>preserving</td>
</tr>
</tbody>
</table>

DepPreserving(X₁,...,Xₖ,F) {
    G := ∅
    for (i=1,..,k)
        G := G U RestrictFDs(Xᵢ,F)
    return IsEquiv(G,F)
}
• Introduction
• Normal Forms
  ▪ BCNF
  ▪ 3NF
• Decomposition
  ▪ NF Decompositions
  ▪ Preserving Data
  ▪ Preserving Dependencies
• Decomposition Algorithms
  ▪ BCNF
  ▪ 3NF
  ▪ 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?

• We discuss 2 decomposition algorithms
  – BCNF decomposition
    • Lossless
  – 3NF decomposition
    • Lossless, dependency preserving, p-time
Outline

• Introduction

• Normal Forms
  ▪ BCNF
  ▪ 3NF

• Decomposition
  ▪ NF Decompositions
  ▪ Preserving Data
  ▪ Preserving Dependencies

• Decomposition Algorithms
  ▪ BCNF
  ▪ 3NF
  ▪ 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 \( X_1,X_2 \subsetneq U \).
- **Proof:**
  - Let \( X \rightarrow Y \) be a BCNF violation (\( X \) is not a superkey and \( Y \) is not a subset of \( X \)).
  - Take \( X_1 = X^+ \) and \( X_2 = X \cup (U \setminus X^+) \).
  - Why are \( X_1 \) and \( X_2 \) strict subsets of \( U \)?
  - Why lossless?
    - Recall the theorem on binary lossless decompositions ...

\[\text{54}\]
BCNF Decomposition

\[
\text{BCNFD}\text{ec}(U,F) \{ \\
\text{if } ((U,F) \text{ in } \text{BCNF}) \\
\qquad \text{return } \{U\} \\
\text{Find a BCNF violation } X \rightarrow Y \\
X_1 := \text{Closure}(X,F) \\
F_1 := \text{RestrictFDs}(X_1,F) \\
X_2 := X \cup (U \setminus X_1) \\
F_2 := \text{RestrictFDs}(X_2,F) \\
\text{return } \text{BCNFD}\text{ec}(X_1,F_1) \cup \text{BCNFD}\text{ec}(X_2,F_2) \\
\}
\]
Execution Example

Are dependencies preserved in this decomposition?

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?

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
  ▪ BCNF
  ▪ 3NF
  ▪ Note on 4NF
We next describe an algorithm for 3NF decomposition

First, some intuition
Intuition

Idea: for dependency preservation, each X→A becomes a schema

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

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

Problem: lossy
Solution: add a key
• Let $F$ be a set of FDs
• A *minimal cover* of $F$ is a set $G$ of FDs with the following properties:
  
  – $G^+ = F^+$
  
  – 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 $G \setminus \{X \rightarrow A\} \models X \rightarrow A$
  
  – All attributes are required: no FD $XB \rightarrow A$ in $G$ is such that $G \models X \rightarrow A$
Revised Example

\{ A \rightarrow B , AB \rightarrow C , C \rightarrow B , D \rightarrow C \}
Algorithm for 3NF Decomposition

3NFDec(U,F) {
    D = Ø
    G := MinCover(F)
    for all (X→A in G) do
        D := D U {XA}
    if (no set in D is a superkey)
        D := D U {FindKey(U,F)}
    D := RemoveConained(D)
    return D
}
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 dependencies are preserved?
Example Revisited

track, faculty → consultant

track → campus

consultant → campus, faculty

faculty → campus

min-cover

track, faculty → consultant

track → campus

consultant → faculty

faculty → campus

removed contained

track, consultant, faculty

track, campus

consultant, faculty

faculty, campus

track, consultant, faculty

track, campus

faculty, campus
Outline

• Introduction

• Normal Forms
  ▪ BCNF
  ▪ 3NF

• Decomposition
  ▪ NF Decompositions
  ▪ Preserving Data
  ▪ Preserving Dependencies

• Decomposition Algorithms
  ▪ BCNF
  ▪ 3NF
  ▪ 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
  - What about dependency preserving?
  - **Answer**: No! Even if there are only FDs (recall BCNF)