Database Management Systems
Faculty of Computer Science
Technion – Israel Institute of Technology
Course 236363
Lecture 7: Schema Normalization

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

Example
- Define the involved attributes
- Determine what dependencies hold in real life
- Decide on desired properties
- Decompose into multiple good (“normalized”) schemas
Notation

- 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 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” deals with the relationship between keys, prime attributes and nonprime attributes

Outline

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

History of Normal Forms

- DB looks like a logical structure; assumed by default
- "Standard" normal form: a nonprime attribute can be determined only by a superkey
- A relation does not involve any "implicit" joins
- No nontrivial FDs except for superkeys
- No nontrivial MVDs except for superkeys

1970
1971
1974
1977
1979

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**:
  - \(\text{name, symbol, dean} \quad \text{BCNF}\)
  - \(\text{name} \rightarrow \text{symbol}\), \(\text{symbol} \rightarrow \text{dean}\), \(\text{dean} \rightarrow \text{name}\)
  - \(\text{follows, followed, fid} \quad \text{BCNF}\)
  - \(\text{follow, followed} \rightarrow \text{fid}\), \(\text{fid} \rightarrow \text{follow, followed}\)

- **Social network**:
  - \(\text{state, city, street, zip} \quad \text{not BCNF}\)
  - \(\text{state, city, street} \rightarrow \text{zip}\), \(\text{zip} \rightarrow \text{state}\)

- **Address**:
  - \(\text{track, faculty, consultant, campus} \quad \text{not BCNF}\)
  - \(\text{track, faculty} \rightarrow \text{consultant}\), \(\text{consultant} \rightarrow \text{faculty}\), \(\text{track} \rightarrow \text{campus}\), \(\text{faculty} \rightarrow \text{campus}\)

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 \(\text{Closure}(X)\)

Outline

- 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**:
  - \(\text{name, symbol, dean} \quad \text{BCNF}\)
  - \(\text{name} \rightarrow \text{symbol}\), \(\text{symbol} \rightarrow \text{dean}\), \(\text{dean} \rightarrow \text{name}\)

- **Social network**:
  - \(\text{follows, followed, fid} \quad \text{BCNF}\)
  - \(\text{follow, followed} \rightarrow \text{fid}\), \(\text{fid} \rightarrow \text{follow, followed}\)

- **Address**:
  - \(\text{state, city, street, zip} \quad \text{not 3NF}\)
  - \(\text{state, city, street} \rightarrow \text{zip}\), \(\text{zip} \rightarrow \text{state}\)

- **Tracks**:
  - \(\text{track, faculty, consultant, campus} \quad \text{not BCNF not 3NF}\)
  - \(\text{track, faculty} \rightarrow \text{consultant}\), \(\text{consultant} \rightarrow \text{faculty}\), \(\text{track} \rightarrow \text{campus}\), \(\text{faculty} \rightarrow \text{campus}\)
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 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 $(F_i)^+[X_i]$
    - That is, each $F_i$ consists of the FDs imposed by $F$ on $X_i$

Decomposing and Composing Relations

- $r$ is decomposed into $r_1, r_2, \ldots, r_k$ and then recombined into $r$
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 $F[X_i]$ can be exponentially larger than $U$
- But this problem is solvable! We can efficiently construct $F_i$’s that satisfy $(F_i) = F[X_i]$

Constructing $F_i$

```plaintext
RestrictFDs(X_i,F) {
    F_i := ∅
    for all (Y → Z in F)
        if (Y ⊆ X_i)
            W := Closure(Y,F) ∩ X_i
            F_i := F_i ∪ {Y → 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 $N$ 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

<table>
<thead>
<tr>
<th>3NF decomposition?</th>
<th>BCNF decomposition?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCD</td>
<td>AD</td>
</tr>
<tr>
<td>A → B, B → C, ABC → D, D → B</td>
<td>A → D, B → C, D → B</td>
</tr>
<tr>
<td>ABCD</td>
<td>AD</td>
</tr>
<tr>
<td>A → B, C → B</td>
<td>A → C</td>
</tr>
<tr>
<td>A → C, C → B</td>
<td>A → C</td>
</tr>
<tr>
<td>A → D, B → C, C → D</td>
<td>A → D, B → C, C → D</td>
</tr>
<tr>
<td>ABCD</td>
<td>AD</td>
</tr>
<tr>
<td>A → B, B → C, C → D</td>
<td>A → B, B → C, C → D</td>
</tr>
<tr>
<td>ABCD</td>
<td>AD</td>
</tr>
<tr>
<td>A → B, B → C, C → D</td>
<td>A → B, B → C, C → D</td>
</tr>
</tbody>
</table>

Outline

- 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 Taub</td>
<td>152</td>
<td></td>
</tr>
<tr>
<td>Amir Meyer</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>Ahuva Meyer</td>
<td>246</td>
<td></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**

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

**Example 2**

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

**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. \( F = X_1 \cap X_2 \rightarrow X_1 \) or \( F = X_1 \cap X_2 \rightarrow X_2 \)
2. \( \{X_1,X_2\} \) is a lossless decomposition

So what would be a decision algorithm in this case?

**Answer:** test whether \( \text{Closure}(F,X_1 \setminus X_2) \) contains either \( X_1 \) or \( X_2 \)
Proof: 1 ⇒ 2

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

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_1 \cap X_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

We know that this is a subset of $r_j$ for some $j$.

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_1 \cap X_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Proof: not 1 ⇒ not 2

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

Let $Y = \{X_1 \cap X_2\}^*$ and suppose that $X_1 \cap Y, X_2 \cap Y$

- Construct a relation $r(1)$ over $U$
  - $|Y| = |Y| = (0, ..., 0)$
  - $|UY| = (1, ..., 1)$ or $|U \cap Y| = (2, ..., 2)$

Claim 1: $r \equiv F$

- Proof similar to completeness of Armstrong’s axioms

Claim 2: $x_j(1) \equiv x_j(1) \neq r$

- The join contains a row with both 1s and 2s

Illustration: not 1 ⇒ not 2

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

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_1 \cap X_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

We know that this is a subset of $r_j$ for some $j$.

The General Case

Losslessness Testing

<table>
<thead>
<tr>
<th>Given:</th>
<th>Goal:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U, F, X_1, ..., X_k$</td>
<td>Determine whether $(X_1, X_2)$ is a lossless decomposition</td>
</tr>
</tbody>
</table>

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

The Idea

We need to prove that $t$ is here!

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

The General Case

Losslessness Testing

<table>
<thead>
<tr>
<th>Given:</th>
<th>Goal:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U, F, X_1, ..., X_k$</td>
<td>Determine whether $(X_1, X_2)$ is a lossless decomposition</td>
</tr>
</tbody>
</table>

1st step: create the “known subset”

- A table over $U$, one tuple $t$, for each $X_j$, $t[A] \neq t$ if $X_j$ contains $A_j$

2nd step: chase

- While the table changes do:
  - Look for an FD violation and equate the conclusions
  - "Equalize" = change every occurrence of one to the other
  - When equating $a_j$ with $y_j$, change it to $A_j$

3rd step: Return true iff there is a row of $a_j$'s
Step 1: construct the known subset

Step 2: chase

Step 3: return true

Think

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

\( T = \{(x_1, f_1), (x_2, f_2), (x_3, f_3), (x_4, f_4)\} \)

\( r \)

\( r_1 \)

\( r_2 \)

\( r_3 \)

\( r_k \)

\( \{x_1, f_1\} , \{x_2, f_2\} , \{x_3, f_3\} , \{x_4, f_4\} \)

Example 2

\( A \rightarrow B, B \rightarrow C, A \rightarrow D, D \rightarrow B \)

\( \{AD, BD, BC\} \)

Are dependencies preserved in this decomposition?

Answer: Yes!

\( A \rightarrow D, B \rightarrow C, D \rightarrow B \)

\( ABCD \)

\{BC, AC\}

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 relations $r_1, ..., r_k$ over $(X_1, F[X_1]), ..., (X_k, F[X_k])$, respectively, $r_i \vdash r_j$ 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 $(X_1, F[X_1]), ..., (X_k, F[X_k])$, respectively, the join $r_1 \Join ... \Join r_k$ satisfies $F$.
2. $F^+ = (F_1 \cup ... \cup F_k)$.

Testing for Dependency Preservation

<table>
<thead>
<tr>
<th>Given:</th>
<th>Goal:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U, F, X_1, ..., X_k$</td>
<td>Determine whether $(X_1, ..., X_k)$ is dependency preserving</td>
</tr>
</tbody>
</table>

```
DepPreserving(X_1, ..., X_k, F) {
  G := ∅
  for (i=1,...,k) {
    G := G \cup RestrictFDs(X_i, F)
  }
  return IsEquiv(G, F)
}
```

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

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 $X \times x^+ \times U$ and $X \times x^+ U \setminus Y^+$.
  - Why are $X_1$ and $X_2$ strict subsets of $U$?
  - Why lossless?
    • Recall the theorem on binary lossless decompositions ...

Outline

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

Outline

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

BCNFDec(U,F) {
    if ((U,F) in BCNF)
      return (U)
    Find a BCNF violation X → Y
    X₁ := Closure(X,Y)
    F₁ := RestrictFDs(X₁,F)
    X₂ := X \ U(F\{X\})
    F₂ := RestrictFDs(X₂,F)
    return BCNFDec(X₁,F₁) U BCNFDec(X₂,F₂)
}

Execution Example

BCNFD = \{AD, BD, BC\}

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?

ABC

AB → C  C → B

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

Algorithm for 3NF Decomposition

• We next describe an algorithm for 3ND decomposition
• First, some intuition
Intuition

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

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

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

Reminder: Minimal Cover

• 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 $G \setminus \{ X \rightarrow A \} \not\models X \rightarrow A$
  – All attributes are required: no FD $XB \rightarrow A$ in $G$ is such that $G \models X \rightarrow A$

Algorithm for 3NF Decomposition

3NFDec($U, F$)

1. $D = \emptyset$
2. $G := \text{MinCover}(F)$
3. for all $(X \rightarrow A)$ in $G$
4. 2. $D := D \cup \{ XA \}$
5. if (no set in $D$ is a superkey)
6. 3. $D := D \cup \{ \text{FindKey}(U, F) \}$
7. $D := \text{RemoveConained}(D)$
8. return $D$

No need for schemas contained in others

Revised Example

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

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

Problem: lossy
Solution: add a key

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
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 → 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 → Y is trivial (always holds) if and only if Y = ∅ or Y = U \ X
- Recall: an FD X → Y can be viewed as a special type of the MVD X → 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 → Y iff for all relations r over U we have:
  \[ r = \pi_X \cup \pi_Y(r) \times \pi_X \cup (U \setminus Y)(r) \]
- Hence, the recursive decomposition algorithm for BCNF decomposition works here
  - Decompose(X \ U) \ U Decompose((X \ U)(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)