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 dependencies hold in real life
2a. Determine implied dependencies
3. Decide on desired properties
4. 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 ⊊ 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
- 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
  - \(Y \subseteq X\)

Can BCNF be Tested Efficiently?

- On the face of it, we need to consider every derived FD (exponentially many); however:
- \textbf{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
  - \textit{But which direction is straightforward?}
  - \textit{So what would be an efficient BCNF testing?}
  
  \textbf{Answer}: For each \(X \rightarrow Y\) in \(F\), test whether \(U = \text{Closure}(F \cup \{X\})\)

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
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 \(F_i = F[X_i]\)
    - That is, each \(F_i\) consists of the FDs imposed by \(F\) on \(X_i\)

Decomposing and Composing Relations
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 $U$.
  - This problem is unavoidable. It may be that $F^+[X_i]$ is not equivalent to any sub-exponential #FDs!
  - We keep this problem in mind - we will not assume that $F[X_i]$ can be instantiated efficiently.

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?

\[
\begin{array}{c}
\text{ABCD} \\
A \rightarrow B, B \rightarrow C, \\
ABC \rightarrow D, D \rightarrow B
\end{array}
\rightarrow
\begin{array}{c}
\text{AD} \\
A \rightarrow D
\end{array}
\rightarrow
\begin{array}{c}
\text{BC} \\
B \rightarrow C
\end{array}
\rightarrow
\begin{array}{c}
\text{BD} \\
D \rightarrow B
\end{array}
\]

Answer: BCNF, 3NF

\[
\begin{array}{c}
\text{ABCD} \\
A \rightarrow B, B \rightarrow C, \\
C \rightarrow B
\end{array}
\rightarrow
\begin{array}{c}
\text{ABC} \\
A \rightarrow C
\end{array}
\rightarrow
\begin{array}{c}
\text{AD} \\
D \rightarrow B
\end{array}
\]

Answer: 3NF, not BCNF

\[
\begin{array}{c}
\text{ABCD} \\
A \rightarrow B, B \rightarrow C, \\
C \rightarrow D
\end{array}
\rightarrow
\begin{array}{c}
\text{ABC} \\
A \rightarrow C
\end{array}
\rightarrow
\begin{array}{c}
\text{ACD} \\
C \rightarrow D
\end{array}
\]

Answer: not 3NF, not BCNF

Outline

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

Good Decomposition?

\[
\begin{array}{|c|c|c|}
\hline
\text{person} & \text{building} & \text{room} \\
\hline
\text{Alma} & \text{Tau} & 152 \\
\text{Amir} & \text{Meyer} & 35 \\
\text{Ahuva} & \text{Meyer} & 246 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{person} & \text{building} & \text{room} \\
\hline
\text{Alma} & \text{Tau} & 152 \\
\text{Amir} & \text{Meyer} & 35 \\
\text{Ahuva} & \text{Meyer} & 246 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{person} & \text{building} & \text{room} \\
\hline
\text{Ahuva} & \text{Meyer} & 35 \\
\text{Ahuva} & \text{Meyer} & 246 \\
\text{Amir} & \text{Meyer} & 152 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{person} & \text{building} & \text{room} \\
\hline
\text{Ahuva} & \text{Meyer} & 246 \\
\text{Amir} & \text{Meyer} & 35 \\
\text{Ahuva} & \text{Meyer} & 152 \\
\hline
\end{array}
\]
The Case of Binary Decomposition

**THEOREM:** Let \( \{X_1, X_2\} \) be a decomposition of \((U,F)\). The following are equivalent:

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

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

*Answer:* test whether \(\text{Closure}(E \times X_1 X_2)\) contains either \(X_1\) or \(X_2\).

---

**Example 2**

<table>
<thead>
<tr>
<th>person</th>
<th>building</th>
<th>room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>Tau b</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>

**Example 1**

<table>
<thead>
<tr>
<th>person</th>
<th>building</th>
<th>room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>Tau b</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>

**Decision Algorithm**

<table>
<thead>
<tr>
<th>Losslessness Testing</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Given:</strong></td>
<td></td>
</tr>
<tr>
<td>( U, F, X_1, \ldots, X_n )</td>
<td></td>
</tr>
<tr>
<td>((X_1, \ldots, X_n)) is a decomposition of ((U,F))</td>
<td></td>
</tr>
<tr>
<td><strong>Determine whether</strong></td>
<td></td>
</tr>
<tr>
<td>((X_1, \ldots, X_n)) is a lossless decomposition</td>
<td></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

---

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

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

---

In any case, we had \( \top \) to begin with... hence lossless
The General Case

Proof: not 1 ⇒ not 2

1. $F = X'R \rightarrow X'$ or $F = X'R \rightarrow X''$
2. $(X,R)$ is a lossless decomposition

- Let $Y = X_1 \cup X_2$ and suppose that $X_1 \subseteq Y$, $X_2 \subseteq Y$
- Construct a relation $r = \{t, u\} \cup U$:
  - $t[Y] = (0, \ldots, 0)$
  - $t[U] = (1, \ldots, 1)$; $u[U] = (2, \ldots, 2)$
- Claim 1: $r \vdash \Pi_{X_i}(r) 
eq \Pi_{X_i}(r)$
  - Proof similar to completeness of Armstrong's axioms
- Claim 2: $\Pi_{X_i}(r) \vdash \Pi_{X_j}(r) 
eq r$
  - The join contains a row with both 1s and 2s

Illustration: not 1 ⇒ not 2

The General Case

Losslessness Testing

Given:
- $U, F, X_1, \ldots, X_n$
- $(X_1, X_2)$ is a decomposition of $(U, F)$

Goal:
Determine whether $(X_1, X_2)$ is a lossless decomposition

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!

Example

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

Example 1

Example 2

Formal Definition

- A decomposition \( X_1, \ldots, X_k \) of \((U,F)\) is dependency preserving if for all relations \( r_1, \ldots, r_k \) over \((X_1,F[X_1]), \ldots, (X_k,F[X_k])\), respectively, \( r_1 \models \cdots \models r_k \) satisfies \( F \)
- Can we test whether a given decomposition has this property?
- THEOREM: The following are equivalent:
  1. For all \( r_1, \ldots, r_k \) over \((X_1,F[X_1]), \ldots, (X_k,F[X_k])\), respectively, the join \( r_1 \bowtie \cdots \bowtie r_k \) satisfies \( F \)
  2. \( F^* = (F[X_1] \cup \cdots \cup F[X_k])^* \)
Testing for Dependency Preservation

- We need to test whether \( F^+ = (F_1 \cup \ldots \cup F_k)^* \)
- \( F^+ \supseteq F_1 \cup \ldots \cup F_k \), so \( F^+ \supseteq (F_1 \cup \ldots \cup F_k)^* \)
- So, need to test whether \( F^+ \subseteq (F_1 \cup \ldots \cup F_k)^* \)
- It suffices to test whether each \( X \rightarrow Y \) in \( F \) is implied by \( F_1 \cup \ldots \cup F_k \)
  - Or in other words, whether \( Y \) is a subset of the closure of \( X \) under \( F_1 \cup \ldots \cup F_k \)
- Next slide: efficient computation of the closure of \( X \) under \( F_1 \cup \ldots \cup F_k \)

Closure w.r.t. a Decomposition

\[
\text{ClosureDecomp}(X,F_1,\ldots,X_k) \{ \\
  Y := X \\
  \text{while}(Y \text{ changes}) \\
  \quad \text{for}(i=1,\ldots,k) \\
  \quad \quad Y := Y \cup (\text{Closure}(Y \cap X_i) \cap X_i) \\
  \text{return } Y 
\}
\]

Outline

- 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
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 \( X_1 = X \) and \( X_2 = U \setminus (U \setminus X) \).
  - Why are \( X_1 \) and \( X_2 \) strict subsets of \( U \)?
  - Why lossless?
    - Recall the theorem on binary lossless decompositions ...

BCNF Decomposition

```
BCNFDec(U,F) {
  if ((U,F) in BCNF)
    return {U}
  Find a BCNF violation \( X \rightarrow Y \)
  \( X_1 := \text{Closure}(X,F) \)
  \( F_1 := F[F[X_1]] \)
  \( X_2 := X \cup (U \setminus X_1) \)
  \( F_2 := F[F[X_2]] \)
  return BCNFDec(X_1,F_1) \cup BCNFDec(X_2,F_2)
}
```

Execution Example

```
ABCD
A \rightarrow B, B \rightarrow C, ABC \rightarrow D, D \rightarrow B

BC
B \rightarrow C

ABD
A \rightarrow BD, D \rightarrow B

BD
D \rightarrow B

AD
A \rightarrow D

\{AD, BD, BC\}
```

Are dependencies preserved in this decomposition?

Answer: Yes, we already saw that previously.

Can Dependencies be Preserved?

```
ABC
AB \rightarrow C, C \rightarrow B

BC
C \rightarrow B

AC

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.

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...

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 3NF decomposition
- First, some intuition

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 \( G \setminus \{ X \rightarrow A \} \models X \rightarrow A \)
  - All attributes are required: no FD \( XB \rightarrow A \) in \( G \) is such that \( G \models X \rightarrow A \)

Algorithm for 3NF Decomposition

\[ 3NFDec(U,F) \{ \\
    D := \emptyset \\
    G := \text{MinCover}(F) \\
    \text{for all } (X \rightarrow A \in G) \text{ do} \\
    \quad D := D \cup \{XA\} \\
    \quad \text{if (no set in } D \text{ is a superkey)} \\
    \quad \quad D := D \cup \{ \text{FindKey}(U,F) \} \\
    \quad D := \text{RemoveConained}(D) \\
    \text{return } D \\
\} \]

Intuition

- Idea: for dependency preservation, each \( X \rightarrow A \) becomes a schema

Revised Example

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

About the Proof

- We will not prove the correctness here
- Still, what needs to be proved?
  - Resulting schemas are all in 3NF
  - Dependencies are preserved
  - Straightforward: all dependencies of the minimal cover are presented
  - Lossless
    - What would the lossless-testing algorithm do when one \( X \) is a key and dependencies are preserved?
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:
  \[
  r = \pi_X (r) \times \pi_Y (r) \]
- Hence, the recursive decomposition algorithm for BCNF decomposition works here.
  - Decompose \((X \cup Y) \cup \text{Decompose}(X \cup (U \setminus Y))\)
  - 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).