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 \(Y \subset K\) 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

Outline

• Introduction
  ▶ Normal Forms
    • BCNF
    • 3NF
• Decomposition
  ▶ NF Decompositions
  ▶ Preserving Data
  ▶ Preserving Dependencies
• Decomposition Algorithms
  • 3NF
  • BCNF
  • 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
- \(Y \subseteq X\)

Examples

- Faculty:
  - follows, followed, fid
  - BCNF
- Social network:
  - follows, followed, fid
  - BCNF
- Address:
  - state, city, street, zip
  - not BCNF
- Tracks:
  - track, faculty, consultant, campus
  - not BCNF

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 \(X \rightarrow Y\) 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:
  - name, symbol, dean
  - BCNF
  - name, symbol, symbol = dean, dean = name
- Social network:
  - follows, followed, fid
  - BCNF
  - follows, followed = fid, fid = follows, followed
- Address:
  - state, city, street, zip
  - not BCNF
  - state, city, street = zip, zip = state
- Tracks:
  - track, faculty, consultant, campus
  - not BCNF
  - track, faculty = consultant, consultant = faculty, track = campus, faculty = campus
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]

### 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 $F_i = \pi_X F$ (i.e., $F_i$ is induced on $X_i$)
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 $F_i^+ [X_i]$ can be expensive, since $F^+$ and $F_i^+ [X_i]$ can be exponentially larger than $U$
  - This problem is unavoidable: It may be that $F_i^+ [X_i]$ is not equivalent to any sub-exponential #FDs!
- We keep this problem in mind – we will not assume that $F_i^+ [X_i]$ can be materialized 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

Outline

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

Examples

3NF decomposition? BCNF decomposition?

<table>
<thead>
<tr>
<th>ABCD</th>
<th>AD</th>
<th>BC</th>
<th>BD</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \rightarrow B, B \rightarrow C, ABC \rightarrow D, D \rightarrow B$</td>
<td>$A \rightarrow D$</td>
<td>$B \rightarrow C$</td>
<td>$D \rightarrow B$</td>
</tr>
<tr>
<td>Answer: BCNF, 3NF</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ABCD</th>
<th>AD</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \rightarrow B, B \rightarrow C, ABC \rightarrow D, D \rightarrow B$</td>
<td>$A \rightarrow C$</td>
</tr>
<tr>
<td>Answer: 3NF, not BCNF</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ABCD</th>
<th>AD</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \rightarrow B, B \rightarrow C, ABC \rightarrow D, D \rightarrow B$</td>
<td>$A \rightarrow D$</td>
</tr>
<tr>
<td>Answer: not 3NF, not BCNF</td>
<td></td>
</tr>
</tbody>
</table>

Good Decomposition?

Can you restore?

person → building, room

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

- Let \( \{X_1, \ldots, X_r\} \) be a decomposition of \( (U,F) \).
- We say that \( \{X_1, \ldots, X_r\} \) is a **lossless decomposition** of \( (U,F) \) if for all relations \( r \) over \( (U,F) \) we have:
  \[ \pi_{X_1}(r) \triangleright \cdots \triangleright \pi_{X_r}(r) = r \]
- Containment in one direction always holds:
  \[ \pi_{X_i}(r) \triangleright \cdots \triangleright \pi_{X_r}(r) \supseteq r \]
- 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>Ahuva</td>
<td>Meyer</td>
<td>152</td>
</tr>
<tr>
<td>Alma</td>
<td>Meyer</td>
<td>246</td>
</tr>
</tbody>
</table>

Example 2

<table>
<thead>
<tr>
<th>Person</th>
<th>Building</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>Meyer</td>
<td>152</td>
</tr>
<tr>
<td>Alma</td>
<td>246</td>
<td></td>
</tr>
</tbody>
</table>

Decision Algorithm

**Losslessness Testing**

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

**Goal:**
- Determine whether \( \{X_1, \ldots, X_r\} \) 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 \( F = (X_2 \cap X_1) \rightarrow X_2 \)
2. \( \{X_1, X_2\} \) is a lossless decomposition

So what would be a decision algorithm in this case?

Proof: 1\( \Rightarrow \)2

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

We need to prove that 1 is here!

In any case, we have:
- to begin with
- Hence lossless!
Proof: not 1 ⇒ not 2
1. \( F \iff X_2 \iff X_1 \) or \( F \iff X_2 \iff X_1 \)
2. \((X_2, X_3)\) is a lossless decomposition

- Let \( X_{12}(X_2 \cap X_1) \) and suppose \( X_2 \nsubseteq X_2 \cdot X_1 \cdot X_2 \)
- Construct a relation \( r(t, u) \) over \( U \):
  - \( X_{12} = \{ X_2 \} \)
  - \( t \in X_{12} \iff t = (0, ..., 0) \)
  - \( u \in X_{13} \iff u = (1, ..., 1) \)
- Claim 1: \( r \vDash F \)
  - Proof similar to completeness of Armstrong's axioms
- Claim 2: \( \pi_{X_2}(r) \nsubseteq \pi_{X_2}(r) \neq r \)
  - The join contains a row with both 1s and 2s

Illustration: not 1 ⇒ not 2
1. \( F \iff X_2 \iff X_1 \) or \( F \iff X_2 \iff X_1 \)
2. \((X_2, X_3)\) is a lossless decomposition

The General Case

Lossless Testing

Given:
- \( U, F, X_1, ..., X_n \)
- \((X_2, X_3)\) is a decomposition of \((U, F)\)

Goal:
- Determine whether \((X_2, X_3)\) is a lossless decomposition

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

The Idea

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

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

The General Case

Lossless Testing

Given:
- \( U, F, X_1, ..., X_n \)
- \((X_2, X_3)\) is a decomposition of \((U, F)\)

Goal:
- Determine whether \((X_2, X_3)\) is a lossless decomposition

- 1st step: create the "known subset"
  - A table over \( U \), one tuple \( t \), for each \( X_i \), \( t[A_i] = a_i \) if \( X_i \) contains \( A_i \)
    and \( t[A_i] \neq a_i \) otherwise

- 2nd step: chase
  - While the table changes do:
    - Look for an FD violation and equate the conclusions
    - "Erase" - change every occurrence of one to the other
      - When replacing \( a_i \) with \( x_i \), change \( x_i \) to \( a_i \)

- 3rd step: Return true iff there is a row of \( a_i \)s
**Step 1:** construct the known subset

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

---

**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**

\[ (U, F) \]

\[ r \]

\[ \{ F_1 \} \]

\[ \{ F_2 \} \]

\[ \{ F_3 \} \]

\[ \{ F_4 \} \]

**Is \( F \) preserved given that each \( F_i \) is preserved in each relation?**

---

**Example 1**

\[ \text{ABCD} \]

\[ A \rightarrow B, B \rightarrow C, ABC \rightarrow D, D \rightarrow B \]

\[ \{ AD, BD, BC \} \]

Are dependencies preserved in this decomposition?

Answer: Yes!

---

**Example 2**

\[ \text{ABC} \]

\[ A \rightarrow B, B \rightarrow C \]

\[ \{ BC, AC \} \]

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 \( r_1, ..., r_k \) over \((X_1,F_1), ..., (X_k,F_k)\), respectively, where \( F_i = F(X_i) \), the relation \( r_i \) satisfies \( F_i \).
- 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 \( r_i \) satisfies \( F_i \).
2. \( F^* = (F_U \cup ...) \) satisfies \( F \).

Testing for Dependency Preservation

- We need to test whether \( F^* = (F_U \cup ...) \) satisfies \( F \).
- \( F^* \supseteq (F_U \cup ...) \), so \( F^* \supseteq (F_U \cup ... \cup F_k) \).
- So, need to test whether \( F^* \supseteq (F_U \cup ...) \).
- It suffices to test whether each \( X \rightarrow Y \) in \( F \) is implied by \( F_U \cup ... \cup F_k \).
- Next slide: efficient computation of the closure of \( X \) under \( F_U \cup ... \cup F_k \).

Without explicitly calculating the \( F_i \)'s!

Closure w.r.t. a Decomposition

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

**Goal:**
Compute the closure of \( X \) under \( F_U \cup ... \cup F_k \).

**ClosureDecomp**

```java
ClosureDecomp(X_i,F_i,X_1,...,X_k) {
    Y := X
    while(Y changes)
        for(i=1,...,k)
            Y := Y \cup (Closure(Y \cap X_i,F_i) \cap X_i)
    return Y
    }
```

Basic claim for \( Z \subseteq X_i \):
- \( Z \subseteq X_i \) if \( Y \subseteq \text{Closure}(Y \cap X_i,F_i) \cap X_i \)

Testing for Dependency Preservation

**Dependency Preservation Testing**

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

**Goal:**
Determine whether \( (X_i, F_i) \) is dependency preserving.

```
DepPreserving(X_1,...,X_k,F) {
    for all \( X \rightarrow Y \) in \( F \)
        if \( Y \subseteq \text{ClosureDecomp}(X_i,F_i,X_1,...,X_k) \)
            return false
    return true
}
```

Outline

- Introduction
- Normal Forms
  - BCNF
  - 3NF
- Decomposition
  - NF Decompositions
  - Preserving Data
  - Preserving Dependencies
- Decomposition Algorithms
  - 3NF
  - BCNF
  - Not covered in the official course material

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

**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 $G \setminus \{X \rightarrow A\} \models X \rightarrow A$
  - All attributes are required: no FD $X \rightarrow A$ in $G$ is such that $G \models X \rightarrow A$
- Exercise: Suggest an algorithm for computing a minimal cover

**Algorithm for 3NF Decomposition**

```
3NFDec(U,F) {
    D = ∅
    G := MinCover(F)
    for all (X→A in G) do
        D := D∪(XA)
        if (no set in D is a superkey)
            D := D∪(FindKey(U,F))
        D := RemoveConained(D)
    return D
}
```

No need for schemas contained in others

**Revised Example**

```
F={A→B, AB→C, C→B, D→C}
```

**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 $X$ is a key and all dependencies preserved?
Example Revisited

<table>
<thead>
<tr>
<th>track</th>
<th>consultant</th>
<th>campus</th>
<th>faculty</th>
</tr>
</thead>
<tbody>
<tr>
<td>track</td>
<td>faculty ⟷ consultant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>track</td>
<td>campuss</td>
<td></td>
<td></td>
</tr>
<tr>
<td>consultant</td>
<td>campus, faculty</td>
<td></td>
<td></td>
</tr>
<tr>
<td>faculty</td>
<td>campus</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 $X_1 = X'$ 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 ...

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[X_1]
    X_2 := X \cup (U \setminus X_1)
    F_2 := F[X_2]
    return BCNFDec(X_1,F_1) \cup BCNFDec(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?

<table>
<thead>
<tr>
<th>ABC</th>
<th>AB → C</th>
<th>C → B</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC</td>
<td>C → B</td>
<td></td>
</tr>
<tr>
<td>AC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 ↠ 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 = ø 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:
  \[ r = \pi_{X \cup Y}(r) \Join \pi_{X \cup (U \setminus Y)}(r) \]
- Hence, the recursive decomposition algorithm for BCNF decomposition works here
  - Decompose(X \cup Y) \Join 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)