3NF

One FD structure causes problems:

- If you decompose, you can’t check the FD’s in the decomposed relations.
- If you don’t decompose, you violate BCNF.

Abstractly: $AB \rightarrow C$ and $C \rightarrow B$.

- In book: title city $\rightarrow$ theatre and theatre $\rightarrow$ city.
- Another example: street city $\rightarrow$ zip, zip $\rightarrow$ city.

Keys: $\{A, B\}$ and $\{A, C\}$, but $C \rightarrow B$ has a left side not a superkey.

- Suggests decomposition into $BC$ and $AC$.
  - But you can’t check the FD $AB \rightarrow C$ in these relations.
Example

\( A = \text{street}, \ B = \text{city}, \ C = \text{zip}. \)

\[
\begin{array}{|c|c|}
\hline
\text{street} & \text{zip} \\
\hline
545 \text{ Tech Sq.} & 02138 \\
545 \text{ Tech Sq.} & 02139 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{city} & \text{zip} \\
\hline
\text{Cambridge} & 02138 \\
\text{Cambridge} & 02139 \\
\hline
\end{array}
\]

Join:

\[
\begin{array}{|c|c|c|}
\hline
\text{city} & \text{street} & \text{zip} \\
\hline
\text{Cambridge} & 545 \text{ Tech Sq.} & 02138 \\
\text{Cambridge} & 545 \text{ Tech Sq.} & 02139 \\
\hline
\end{array}
\]
“Elegant” Workaround

Define the problem away.

- A relation $R$ is in 3NF iff for every nontrivial FD $X \rightarrow A$, either:
  
  1. $X$ is a superkey, or
  2. $A$ is prime = member of at least one key.

- Thus, the canonical problem goes away: you don’t have to decompose because all attributes are prime and therefore no 3NF violations can occur.
Taking Advantage of 3NF

**Theorem:** For any relation $R$ and set of FD’s $F$, we can find a decomposition of $R$ into 3NF relations, such that if the decomposed relations satisfy their projected dependencies from $F$, then their join will satisfy $F$ itself.

- In fact, with some more effort, we can guarantee that the decomposition is also “lossless”; i.e., the join of the projections of $R$ onto the decomposed relations is always $R$ itself, just as for the BCNF decomposition.

- But what we give up is absolute absence of redundancy due to FD’s.

- The “obvious” approach of doing a BCNF decomposition, but stopping when a relation schema is in 3NF, doesn’t always work — it might still allow some FD’s to get lost.
Roadmap

1. Study *minimal* sets of FD’s: needed for the decompositions.
   ✦ Requires study of when two sets of FD’s are *equivalent*, in the sense that they are satisfied by exactly the same relation instances.

2. Give the algorithm for constructing a decomposition into 3NF schemas that preserves all FD’s.
   ✦ Called the *synthesis* algorithm.

3. Show how to modify this construction to guarantee losslessness.
**3NF Synthesis Algorithm**

Roughly, we create for each FD in $F$ a relation containing only its attributes.

- But: we need first to make $F$ *minimal* in the sense that:
  
  a) No FD can be eliminated from $F$.
  
  b) No attribute can be eliminated from a left side of an FD of $F$.

- Note that minimal sets of FD’s are not necessarily unique.

**Equivalent Sets of FD’s**

FD sets are *equivalent* if they each derive the other, i.e., if they allow the same set of relation instances.

- For each of (a) and (b) in the definition of “minimality,” we mean “without making a set of FD’s inequivalent to $F$.”
Testing Equivalence

\[ X_1 \rightarrow A_1 \quad Y_1 \rightarrow B_1 \]
\[ X_2 \rightarrow A_2 \quad Y_2 \rightarrow B_2 \]
\[ \ldots \quad \ldots \]
\[ X_n \rightarrow A_n \quad Y_m \rightarrow B_m \]

- For each \( i \), \( Y_i \rightarrow B_i \) must follow from the set on the left.
  - i.e, \((Y_i)^+\) must contain \( B_i \), when closure is computed using the FD’s on the left.

- Also, each \( X_i \rightarrow A_i \) must follow from the set on the right.

- Important special case: no need to check an FD that appears in both sets.
Example

Suppose $F$ has $A \rightarrow B$, $B \rightarrow C$, and $AC \rightarrow D$.

- $F$ is not minimal.
- $F_1$ with $A \rightarrow B$, $B \rightarrow C$, and $A \rightarrow D$ is minimal.
  - Note that from $F$ we can infer $A \rightarrow D$, and from $F_1$ we can infer $AC \rightarrow D$.
- $F_2$ consisting of $A \rightarrow B$, $B \rightarrow C$ and $C \rightarrow D$ is not equivalent to $F$.
  - Note you cannot infer $C \rightarrow D$ from $F$. 
A Dependency-Preserving Decomposition

1. Minimize the given set of dependencies.

2. Create a relation with schema $XY$ for each FD $X \rightarrow Y$.

3. Eliminate a relation schema that is a subset of another.

4. Add in a relation schema with all attributes that are not part of any FD.
Example

- Start with $R = ABCD$ and $F$ consisting of $A \rightarrow B$, $B \rightarrow C$, and $AC \rightarrow D$.
- $F1$ with $A \rightarrow B$, $B \rightarrow C$, and $A \rightarrow D$ is a minimal equivalent.
- With $F1$ as our minimal set of FD’s, we get database schema $AB$, $BC$, and $AD$, which is sufficient to check $F1$ and therefore $F$. 
Dependency Preservation with Losslessness

Same as for just dependency preservation, but add in a relation schema consisting of a key for $R$.

Example

In above example, $A$ is a key for $R$, so we should add $A$ as a relation schema. However, $A$ is a subset of $AB$, and so nothing is needed; the original database schema $\{AB, BC, AD\}$ is lossless.

Not Covered

- Why basing the decomposition on a minimal equivalent set of FD’s guarantees 3NF.
- Why the key + FD’s synthesis approach guarantees losslessness.
Multivalued Dependencies

Consider the relation \texttt{Drinkers(name, addr, phone, beersLiked)}, with the FD \texttt{name → addr}. That is, drinkers can have several phones and like several beers. Typical relation:

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>phone</th>
<th>beersLiked</th>
</tr>
</thead>
<tbody>
<tr>
<td>joe</td>
<td>a</td>
<td>p1</td>
<td>b1</td>
</tr>
<tr>
<td>joe</td>
<td>a</td>
<td>p1</td>
<td>b2</td>
</tr>
<tr>
<td>joe</td>
<td>a</td>
<td>p1</td>
<td>b3</td>
</tr>
<tr>
<td>joe</td>
<td>a</td>
<td>p2</td>
<td>b1</td>
</tr>
<tr>
<td>joe</td>
<td>a</td>
<td>p2</td>
<td>b2</td>
</tr>
<tr>
<td>joe</td>
<td>a</td>
<td>p2</td>
<td>b3</td>
</tr>
</tbody>
</table>

- Key = \{name, phone, beersLiked\}.
- BCNF violation: \texttt{name → addr}. Decompose into \texttt{D1(name, addr)}, \texttt{D2(name, phone, beersLiked)}.
  - Both are in BCNF.
• But look at $D2$:

<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
<th>beersLiked</th>
</tr>
</thead>
<tbody>
<tr>
<td>joe</td>
<td>$p1$</td>
<td>$b1$</td>
</tr>
<tr>
<td>joe</td>
<td>$p1$</td>
<td>$b2$</td>
</tr>
<tr>
<td>joe</td>
<td>$p1$</td>
<td>$b3$</td>
</tr>
<tr>
<td>joe</td>
<td>$p2$</td>
<td>$b1$</td>
</tr>
<tr>
<td>joe</td>
<td>$p2$</td>
<td>$b2$</td>
</tr>
<tr>
<td>joe</td>
<td>$p2$</td>
<td>$b3$</td>
</tr>
</tbody>
</table>

• The phones and beers are each repeated.

✦ If Joe had $n$ phones and liked $m$ beers, there would be $nm$ tuples for Joe, when $\max(n, m)$ should be enough.
**Multivalued Dependencies**

The *multivalued dependency* $X \rightarrow Y$ holds in a relation $R$ if whenever we have two tuples of $R$ that agree in all the attributes of $X$, then we can swap their $Y$ components and get two new tuples that are also in $R$.

$$
\begin{array}{ccc}
X & Y & \text{others} \\
\hline
\hline \\
\end{array}
$$

![Diagram of Multivalued Dependency]

\[ 
\begin{array}{c}
\text{X} \\
\text{Y} \\
\text{others} \\
\end{array}
\]
**Example**

In **Drinkers**, we have MVD $\text{name} \rightarrow \text{phone}$. For example:

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>phone</th>
<th>beersLiked</th>
</tr>
</thead>
<tbody>
<tr>
<td>joe</td>
<td>$a$</td>
<td>$p_1$</td>
<td>$b_1$</td>
</tr>
<tr>
<td>joe</td>
<td>$a$</td>
<td>$p_2$</td>
<td>$b_2$</td>
</tr>
</tbody>
</table>

with *phone* components swapped yields:

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>phone</th>
<th>beersLiked</th>
</tr>
</thead>
<tbody>
<tr>
<td>joe</td>
<td>$a$</td>
<td>$p_1$</td>
<td>$b_2$</td>
</tr>
<tr>
<td>joe</td>
<td>$a$</td>
<td>$p_2$</td>
<td>$b_1$</td>
</tr>
</tbody>
</table>

which are also tuples of the relation.

- **Note:** we must check this condition for *all* pairs of tuples that agree on *name*, not just one pair.
MVD Rules

1. Every FD is an MVD.
   ✦ Because if $X \rightarrow Y$, then swapping $Y$’s doesn’t create new tuples.
   ✦ Example, in Drinkers: name $\rightarrow$ addr.

2. Complementation: if $X \rightarrow Y$, then $X \rightarrow Z$, where $Z$ is all attributes not in $X$ or $Y$.
   ✦ Example: since name $\rightarrow$ phone holds in Drinkers, so does name $\rightarrow$ addr beersLiked.
Splitting Doesn’t Hold

Sometimes you need to have several attributes on the left of an MVD. For example:

\[ \text{Drinkers(name, areaCode, phone, beersLiked, beerManf)} \]

<table>
<thead>
<tr>
<th>name</th>
<th>areaCode</th>
<th>phone</th>
<th>BeersLiked</th>
<th>beerManf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>650</td>
<td>555-1111</td>
<td>Bud</td>
<td>A.B.</td>
</tr>
<tr>
<td>Joe</td>
<td>650</td>
<td>555-1111</td>
<td>WickedAle</td>
<td>Pete’s</td>
</tr>
<tr>
<td>Joe</td>
<td>415</td>
<td>555-9999</td>
<td>Bud</td>
<td>A.B.</td>
</tr>
<tr>
<td>Joe</td>
<td>415</td>
<td>555-9999</td>
<td>WickedAle</td>
<td>Pete’s</td>
</tr>
</tbody>
</table>

- name → areaCode phone holds, but neither name → areaCode nor name → phone do.
4NF

Eliminate redundancy due to multiplicative effect of MVD’s.

- Roughly: treat MVD’s as FD’s for decomposition, but not for finding keys.
- Formally: \( R \) is in Fourth Normal Form if whenever MVD \( X \rightarrow Y \) is nontrivial (\( Y \) is not a subset of \( X \), and \( X \cup Y \) is not all attributes), then \( X \) is a superkey.
  
  ✦ Remember, \( X \rightarrow Y \) implies \( X \rightarrow\rightarrow Y \), so 4NF is more stringent than BCNF.

- Decompose \( R \), using 4NF violation \( X \rightarrow Y \), into \( XY \) and \( X \cup (R - Y) \).
Example

Drinkers(name, addr, areaCode, phone, beersLiked, beerManf)

- FD: name → addr
- Nontrivial MVD’s: name →→ areaCode phone and name →→ beersLiked beerManf.
- Only key: \{name, areaCode, phone, beersLiked, beerManf\}
- All three dependencies violate 4NF.
- Successive decomposition yields 4NF relations:
  D1(name, addr)
  D2(name, areaCode, phone)
  D3(name, beersLiked, beerManf)