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: $AB \to C$ and $C \to B$.

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

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

- Suggests decomposition into BC and AC.
  - ❖ But you can't check the FD $AB \to C$ in these relations.

Example

$A = \texttt{street}$, $B = \texttt{city}$, $C = \texttt{zip}$.

street	zip
545 Tech Sq.	02138
545 Tech Sq.	02139

city	zip
Cambridge	02138
Cambridge	02139

Join:

city	street	zip
Cambridge	545 Tech Sq.	02138
Cambridge	545 Tech Sq.	02139

“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.

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

Goal: decompose a relation R with FD's F so all relations are 3NF, yet are capable of checking F.

- Roughly, we create for each FD in F a relation containing only its attributes.
- Exception: it is a good idea to merge common left sides; i.e., if $X \rightarrow Y$ and $X \rightarrow Z$ are FD's, make one relation $X \rightarrow YZ$.
- But — we need first to make F *minimal* in the sense that:

  a) No FD can be eliminated from F, and

  b) No attribute can be eliminated from a left side of an FD of F

  without producing a set of FD's that is not equivalent to F.
- Note that minimal sets of FD's are not necessarily unique.

Why is Minimality Important?

Example: $A \to B$, $B \to C$, and $AB \to C$.

- Tells us to create a relation ABC.
  - But that's not in 3NF because A is the only key, and $B \to C$ holds.

Testing Equivalence of FD Sets

Check whether each FD follows logically from the other set.

$$\begin{array}{ll} X1 \rightarrow A1 & Y1 \rightarrow B1 \\ X2 \rightarrow A2 & Y2 \rightarrow B2 \\ \ldots & \ldots \\ Xn \rightarrow An & Ym \rightarrow Bm \end{array}$$

- For each i, $Yi \rightarrow Bi$ must follow from the set on the left.
  - i.e, $(Yi)^+$ must contain Bi, when closure is computed using the FD's *on the left.*
- Also, each $Xi \rightarrow Ai$ must follow from the set on the right.

- Important special case: no need to check an FD that appears in both sets.
- Thus, if F' is constructed from F by deleting an FD, all we have to check is that the deleted FD follows from F'.
- If F' is constructed from F by deleting some attributes from the left side of one FD (i.e., F has $XY \rightarrow Z$ and F' has $X \rightarrow Z$), then:
  - ❖ Surely $XY \rightarrow Z$ follows from $X \rightarrow Z$.
  - ❖ Thus, to check $F \equiv F'$, we have only to check that $X \rightarrow Z$ follows from all of F, i.e., Z is in X^+ when the closure is computed with respect to F.

Example

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

- F is not minimal.
- $F1$ with $A \to B$, $B \to C$, and $A \to D$ is minimal.
  - Note that from F we can infer $A \to D$, and from $F1$ we can infer $AC \to D$.
- $F2$ consisting of $A \to B$, $B \to C$ and $C \to D$ is *not* equivalent to F.
  - Note you cannot infer $C \to D$ 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 `Drinkers(name, addr, phone, beersLiked)`, with the FD `name` $\rightarrow$ `addr`. That is, drinkers can have several phones and like several beers. Typical relation:

name	addr	phone	beersLiked
sue	a	$p1$	$b1$
sue	a	$p1$	$b2$
sue	a	$p1$	$b3$
sue	a	$p2$	$b1$
sue	a	$p2$	$b2$
sue	a	$p2$	$b3$

- Key = {`name, phone, beersLiked`}.
- BCNF violation: `name` $\rightarrow$ `addr`. Decompose into `D1(name, addr)`, `D2(name, phone, beersLiked)`.
  - Both are in BCNF.

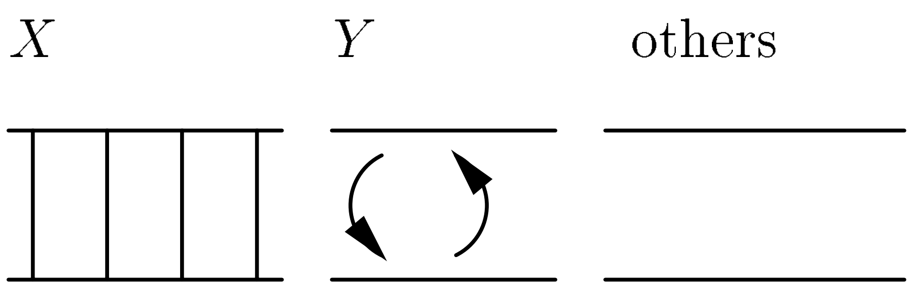
- But look at $D2$:

name	phone	beersLiked
sue	$p1$	$b1$
sue	$p1$	$b2$
sue	$p1$	$b3$
sue	$p2$	$b1$
sue	$p2$	$b2$
sue	$p2$	$b3$

- The phones and beers are each repeated.
  - If Sue had n phones and liked m beers, there would be nm tuples for Sue, when $\max(n, m)$ should be enough.

Multivalued Dependencies

The *multivalued dependency* $X \twoheadrightarrow Y$ 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.



Example

In `Drinkers`, we have MVD `name` $\twoheadrightarrow$ `phone`. For example:

name	addr	phone	beersLiked
sue	a	$p1$	$b1$
sue	a	$p2$	$b2$

with `phone` components swapped yields:

name	addr	phone	beersLiked
sue	a	$p1$	$b2$
sue	a	$p2$	$b1$

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 between tuples that agree on X doesn't create new tuples.

   - Example, in `Drinkers`: `name` $\rightarrow\!\!\!\rightarrow$ `addr`.

2. *Complementation*: if $X \rightarrow\!\!\!\rightarrow Y$, then $X \rightarrow\!\!\!\rightarrow Z$, where Z is all attributes not in X or Y.

   - Example: since `name` $\rightarrow\!\!\!\rightarrow$ `phone` holds in `Drinkers`, so does `name` $\rightarrow\!\!\!\rightarrow$ `addr beersLiked`.

Splitting Doesn't Hold

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

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

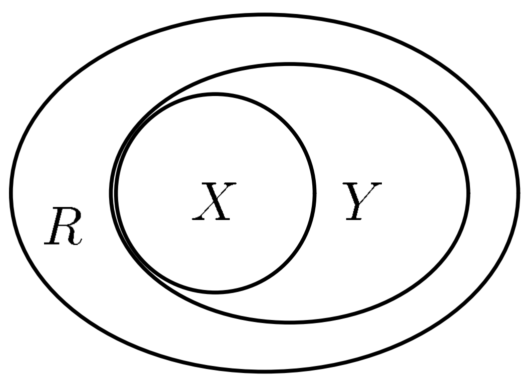
name	areaCode	phone	BeersLiked	beerManf
Sue	650	555-1111	Bud	A.B.
Sue	650	555-1111	WickedAle	Pete's
Sue	415	555-9999	Bud	A.B.
Sue	415	555-9999	WickedAle	Pete's

- `name` $\twoheadrightarrow$ `areaCode phone` holds, but neither `name` $\twoheadrightarrow$ `areaCode` nor `name` $\twoheadrightarrow$ `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 $X \twoheadrightarrow Y$ 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 $X \twoheadrightarrow Y$, so 4NF is more stringent than BCNF.
- Decompose R, using 4NF violation $X \twoheadrightarrow Y$, into XY and $X \cup (R - Y)$.



Example

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

- FD: `name` $\rightarrow$ `addr`
- Nontrivial MVD's: `name` $\rightarrow\!\!\!\rightarrow$ `areaCode phone` and `name` $\rightarrow\!\!\!\rightarrow$ `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)
```