Inferring FD's

And this is important because . . .

- When we talk about improving relational designs, we often need to ask "does this FD hold in this relation?"

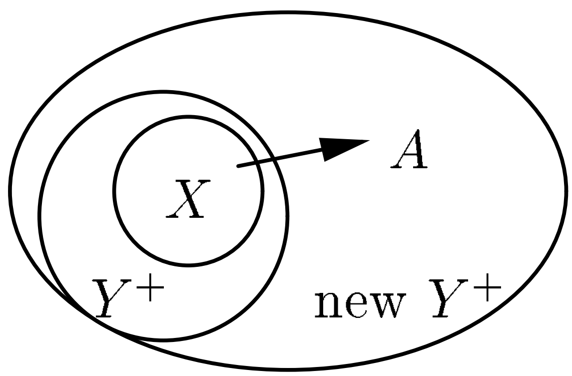
Given FD's $X1 \rightarrow A1$, $X2 \rightarrow A2 \cdots Xn \rightarrow An$, does FD $Y \rightarrow B$ necessarily hold in the same relation?

- Start by assuming two tuples agree in Y. Use given FD's to infer other attributes on which they must agree. If B is among them, then yes, else no.

Algorithm

Define Y^+ = *closure* of Y = set of attributes functionally determined by Y:

- Basis: $Y^+ := Y$.
- Induction: If $X \subseteq Y^+$, and $X \rightarrow A$ is a given FD, then add A to Y^+.

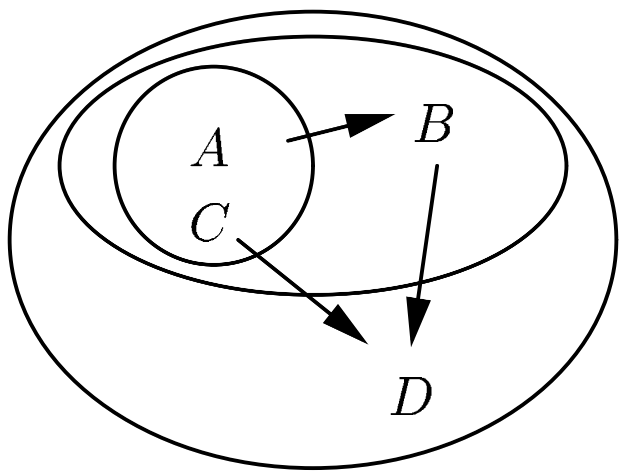


- End when Y^+ cannot be changed.

Example

$A \rightarrow B,\ BC \rightarrow D.$

- $A^+ = AB.$
- $C^+ = C.$
- $(AC)^+ = ABCD.$



Given Versus Implied FD's

Typically, we state a few FD's that are known to hold for a relation R.

- Other FD's may follow logically from the given FD's; these are *implied FD's*.
- We are free to choose any *basis* for the FD's of R — a set of FD's that imply all the FD's that hold for R.

Finding All Implied FD's

Motivation: Suppose we have a relation $ABCD$ with some FD's F. If we decide to decompose $ABCD$ into ABC and AD, what are the FD's for ABC, AD?

- Example: $F = AB \rightarrow C$, $C \rightarrow D$, $D \rightarrow A$. It looks like just $AB \rightarrow C$ holds in ABC, but in fact $C \rightarrow A$ follows from F and applies to relation ABC.

- Problem is exponential in worst case.

Algorithm

For each set of attributes X compute X^+.

- Add $X \rightarrow A$ for each A in $X^+ - X$.
- Ignore or drop some "obvious" dependencies that follow from others:

1. *Trivial FD's*: right side is a subset of left side.
   - ✦ Consequence: no point in computing $\emptyset^+$ or closure of full set of attributes.
2. Drop $XY \rightarrow A$ if $X \rightarrow A$ holds.
   - ✦ Consequence: If X^+ is all attributes, then there is no point in computing closure of supersets of X.
3. Ignore FD's whose right sides are not single attributes.

- Notice that after we project the discovered FD's onto some relation, the FD's eliminated by rules 1, 2, and 3 can be inferred *in the projected relation.*

Example

Example: $F = AB \rightarrow C, C \rightarrow D, D \rightarrow A$. What FD's follow?

- $A^+ = A$; $B^+ = B$ (nothing).
- $C^+ = ACD$ (add $C \rightarrow A$).
- $D^+ = AD$ (nothing new).
- $(AB)^+ = ABCD$ (add $AB \rightarrow D$; skip all supersets of AB).
- $(BC)^+ = ABCD$ (nothing new; skip all supersets of BC).
- $(BD)^+ = ABCD$ (add $BD \rightarrow C$; skip all supersets of BD).
- $(AC)^+ = ACD$; $(AD)^+ = AD$; $(CD)^+ = ACD$ (nothing new).
- $(ACD)^+ = ACD$ (nothing new).
- All other sets contain AB, BC, or BD, so skip.
- Thus, the only interesting FD's that follow from F are: $C \rightarrow A$, $AB \rightarrow D$, $BD \rightarrow C$.

Normalization

Goal = BCNF = Boyce-Codd Normal Form = all FD's follow from the fact "key $\rightarrow$ everything."

- Formally, R is in BCNF if every nontrivial FD for R, say $X \rightarrow A$, has X a superkey.
  - ✦ "Nontrivial" = right-side attribute not in left side.

Why?

1. Guarantees no redundancy due to FD's.
2. Guarantees no *update anomalies* = one occurrence of a fact is updated, not all.
3. Guarantees no *deletion anomalies* = valid fact is lost when tuple is deleted.

Example of Problems

`Drinkers(name, addr, beersLiked, manf, favoriteBeer)`

name	addr	beersLiked	manf	favoriteBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	???	WickedAle	Pete's	???
Spock	Enterprise	Bud	???	Bud

FD's:

1. `name` → `addr`
2. `name` → `favoriteBeer`
3. `beersLiked` → `manf`

- ???'s are redundant, since we can figure them out from the FD's.
- Update anomalies: If Janeway gets transferred to the *Intrepid*, will we change `addr` in each of her tuples?
- Deletion anomalies: If nobody likes Bud, we lose track of Bud's manufacturer.

Each of the given FD's is a BCNF violation:

- Key = {`name`, `beersLiked`}
  - ✦ Each of the given FD's has a left side a proper subset of the key.

Another Example

`Beers(`<u>`name`</u>`, manf, manfAddr)`.

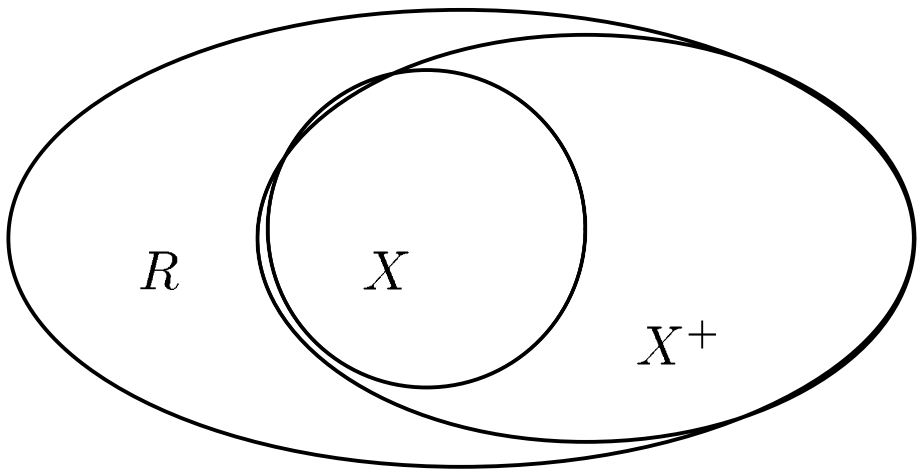
- FD's = `name` $\rightarrow$ `manf`, `manf` $\rightarrow$ `manfAddr`.
- Only key is `name`.
  - ✦ `manf` $\rightarrow$ `manfAddr` violates BCNF with a left side unrelated to any key.

Decomposition to Reach BCNF

Setting: relation R, given FD's F. Suppose relation R has BCNF violation $X \rightarrow B$.

- We need only look among FD's of F for a BCNF violation.
- Proof: If $Y \rightarrow A$ is a BCNF violation and follows from F, then the computation of Y^+ used at least one FD $X \rightarrow B$ from F.
  - ✦ X must be a subset of Y.
  - ✦ Thus, if Y is not a superkey, X cannot be a superkey either, and $X \rightarrow B$ is also a BCNF violation.

1. Compute X^+.
   - ✦ Cannot be all attributes — why?
2. Decompose R into X^+ and $(R - X^+) \cup X$.



3. Find the FD's for the decomposed relations.
   - ✦ Project the FD's from F = calculate all consequents of F that involve only attributes from X^+ or only from $(R - X^+) \cup X$.

Example

$R =$ `Drinkers(`<u>`name`</u>`, addr, `<u>`beersLiked`</u>`, manf, favoriteBeer)`

$F =$

1. `name` $\rightarrow$ `addr`
2. `name` $\rightarrow$ `favoriteBeer`
3. `beersLiked` $\rightarrow$ `manf`

Pick BCNF violation `name` $\rightarrow$ `addr`.

- Close the left side: $\texttt{name}^+ =$ `name addr favoriteBeer`.
- Decomposed relations:

  `Drinkers1(`<u>`name`</u>`, addr, favoriteBeer)`
  `Drinkers2(`<u>`name`</u>`, `<u>`beersLiked`</u>`, manf)`

- Projected FD's (skipping a lot of work that leads nowhere interesting):
  - For `Drinkers1`: `name` $\rightarrow$ `addr` and `name` $\rightarrow$ `favoriteBeer`.
  - For `Drinkers2`: `beersLiked` $\rightarrow$ `manf`.

- BCNF violations?
  - ✦ For `Drinkers1`, `name` is key and all left sides of FD's are superkeys.
  - ✦ For `Drinkers2`, `{name, beersLiked}` is the key, and `beersLiked` $\rightarrow$ `manf` violates BCNF.

Decompose `Drinkers2`

- Close `beersLiked`$^+$ = `beersLiked, manf`.
- Decompose:

  Drinkers3(<u>beersLiked</u>, manf)
  Drinkers4(<u>name</u>, <u>beersLiked</u>)

- Resulting relations are all in BCNF:

  Drinkers1(<u>name</u>, addr, favoriteBeer)
  Drinkers3(<u>beersLiked</u>, manf)
  Drinkers4(<u>name</u>, <u>beersLiked</u>)

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

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

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.

What 3NF Gives You

There are two important properties of a decomposition:

1. We should be able to recover from the decomposed relations the data of the original.

   - Recovery involves projection and join, which we shall defer until we've discussed relational algebra.

2. We should be able to check that the FD's for the original relation are satisfied by checking the projections of those FD's in the decomposed relations.

- Without proof, we assert that it is always possible to decompose into BCNF and satisfy (1).
- Also without proof, we can decompose into 3NF and satisfy both (1) and (2).
- But it is not possible to decompose into BNCF and get both (1) and (2).

  - Street-city-zip is an example of this point.