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^+.

- Eliminate 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. Eliminate $XY \rightarrow Z$ if $X \rightarrow Z$ holds.

   ❖ Consequence: If X^+ is all attributes, then there is no point in computing closure of supersets of X.

3. Eliminate FD's whose right sides are not single attributes.

Example

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

- $A^+ = A$; $B^+ = B$ (nothing).
- $C^+ = ACD$ (add $C \to A$).
- $D^+ = AD$ (nothing new).
- $(AB)^+ = ABCD$ (add $AB \to D$; skip all supersets of AB).
- $(BC)^+ = ABCD$ (nothing new; skip all supersets of BC).
- $(BD)^+ = ABCD$ (add $BD \to C$; 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: $C \to A$, $AB \to D$, $BD \to C$.

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.

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` $\rightarrow$ `addr`
2. `name` $\rightarrow$ `favoriteBeer`
3. `beersLiked` $\rightarrow$ `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(name, 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

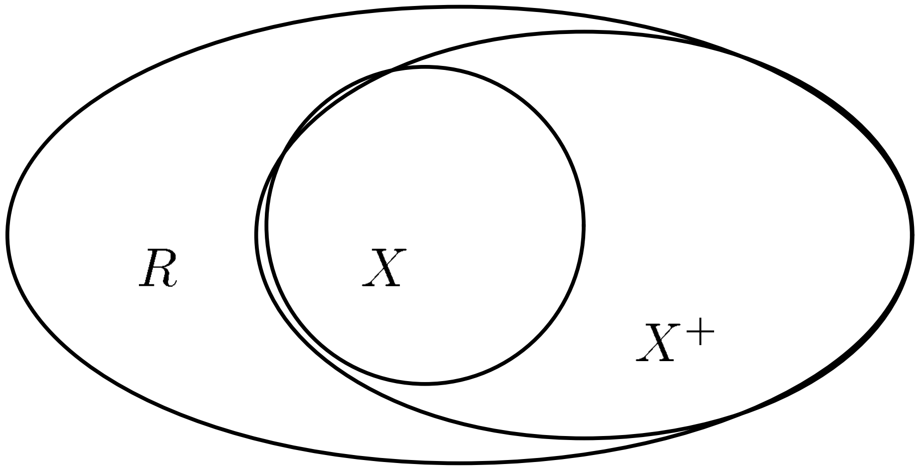
Setting: relation R, given FD's F. Suppose relation R has BCNF violation $X \rightarrow A$.

- Notice: we need only look among FD's of F, because any nontrivial FD that follows from them must contain one of their left sides in its left side.
  - Thus, any FD that follows and has a non-superkey as a left side means there is an FD in F with the same property.

1. *Expand* right side to include 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(name, addr, beersLiked, manf, favoriteBeer)`

$F =$

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

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

- Expand right side:

  `name` $\rightarrow$ `addr favoriteBeer`.

- Decomposed relations:

  `Drinkers1(name, addr, favoriteBeer)`
  `Drinkers2(name, beersLiked, manf)`

- Projected FD's (skipping a lot of work that leads nowehere 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 are superkeys.
  - ❖ For `Drinkers2`, `{name, beersLiked}` is the key, and `beersLiked` → `manf` violates BCNF.

Decompose `Drinkers2`

- Expand: nothing.
- Decompose:

```
Drinkers3(beersLiked, manf)
Drinkers4(name, beersLiked)
```

- Resulting relations are all in BCNF:

```
Drinkers1(name, addr, favoriteBeer)
Drinkers3(beersLiked, manf)
Drinkers4(name, beersLiked)
```

Why Decomposition "Works"?

What does it mean to "work"? Why can't we just tear sets of attributes apart as we like?

- Answer: the decomposed relations need to represent the same information as the original.

Projection and Join

- The operations that relate original and decomposed relations.
- Suppose R is decomposed into S and T. We *project* R onto S by:
    1. Eliminate columns of R not in S.
    2. Eliminate duplicate rows.

Example

$R =$

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

- Project onto `Drinkers1(name, addr, favoriteBeer)`:

name	addr	favoriteBeer
Janeway	Voyager	WickedAle
Spock	Enterprise	Bud

- Project onto `Drinkers3(beersLiked, manf)`:

beersLiked	manf
Bud	A.B.
WickedAle	Pete's

- Project onto `Drinkers4(name, beersLiked)`:

name	beersLiked
Janeway	Bud
Janeway	WickedAle
Spock	Bud

Reconstruction of Original

Can we figure out the original relation from the decomposed relations?

- Sometimes, if we (natural) *join* the relations.
- $R \bowtie S$:
  - ❖ Schema = union of attributes of R and S.
  - ❖ Tuples = all formed from a tuple r from R and s from S that agree in all common attributes.

Example

`Drinkers3` $\bowtie$ `Drinkers4` =

name	beersLiked	manf
Janeway	Bud	A.B.
Janeway	WickedAle	Pete's
Spock	Bud	A.B.

- Join of above with `Drinkers1` = original R.

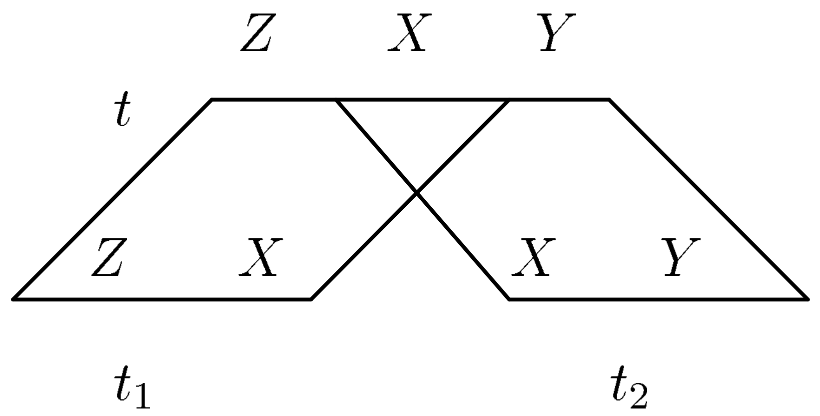
Theorem

Suppose we decompose a relation with schema XYZ into XY and XZ and project the relation for XYZ onto XY and XZ. Then $XY \bowtie XZ$ is *guaranteed* to reconstruct XYZ if and only if either $X \to Y$ or $X \to Z$ holds.

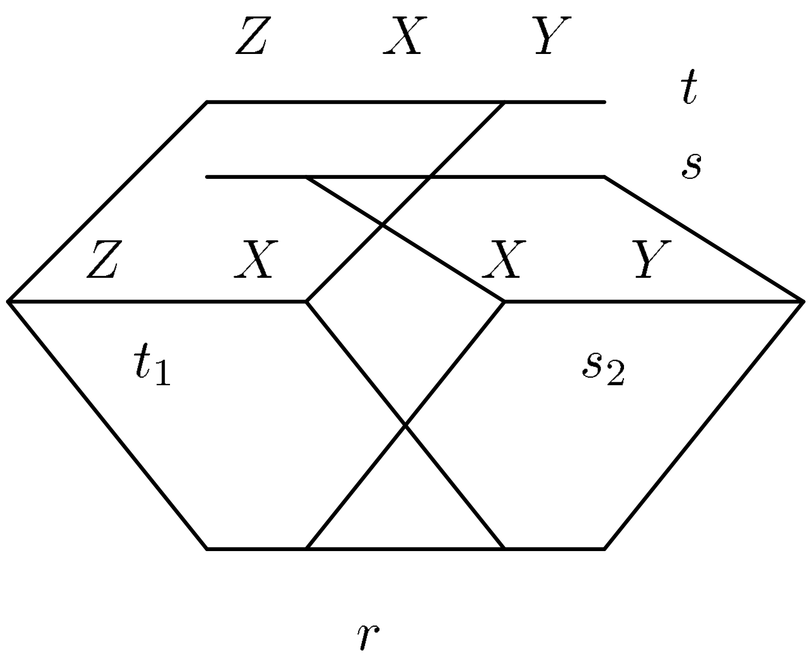
- Notice that whenever we decompose because of a BCNF violation, one of these FD's must hold.

Proof (if)

1. Anything you project comes back in the join.
   - Doesn't depend on FD's.



2. Anything that comes back in the join was in the original XYZ.



- Notice that t_1 and s_2 agree on X.
- If $X \rightarrow Y$, then $r = t$.
- If $X \rightarrow Z$, then $r = s$.
- Either way, r is in original XYZ.

Proof (only-if)

If neither $X \to Y$ nor $X \to Z$ holds, then we can find an example XYZ relation where the project-join returns too much.

Z	X	Y
$z1$	x	$y1$
$z2$	x	$y2$

Z	X
$z1$	x
$z2$	x

X	Y
x	$y1$
x	$y2$

Z	X	Y
$z1$	x	$y1$
$z1$	x	$y2$
$z2$	x	$y1$
$z2$	x	$y2$