

Chapter 3 Design Theory for Relational Databases

Contents

- Functional Dependencies
- Decompositions
- Normal Forms (BCNF, 3NF)
- Multivalued Dependencies (and 4NF)
- Reasoning About FD's + MVD's

Our example of chapter 2

Beers(name, manf)

Bars(name, addr, license)

Drinkers(name, addr, phone)

Likes(drinker, beer)

Sells(bar, beer, price)

Frequents(drinker, bar)

Some questions:

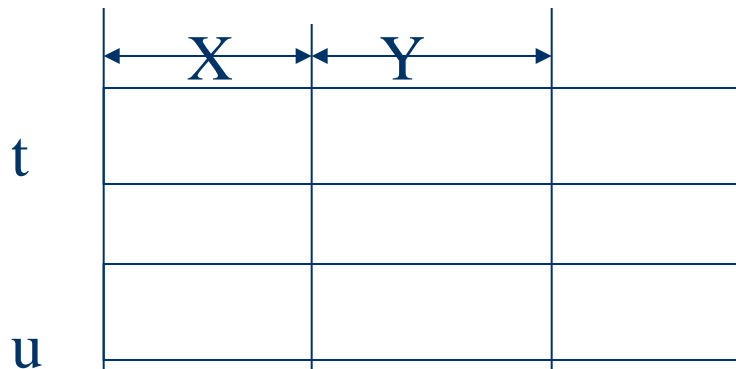
1. Why do we design relations like the example?
2. What makes a good relational database schema?
3. what we can do if it has flaws?

Functional Dependencies

- $X \rightarrow Y$ is an **assertion** about a **relation R** that whenever two tuples of R agree on all the attributes of X , then they must also agree on all attributes in set Y .
 - Say " $X \rightarrow Y$ holds in R ."
 - Convention**: ..., X, Y, Z represent **sets of attributes**; A, B, C, \dots represent **single attributes**.
 - Convention**: no set formers in sets of attributes, just ABC , rather than $\{A, B, C\}$.

Functional Dependency (cont.)

- Exist in a **relational schema** as a **constraint**.
- Agree for all **instances** of the schema. (*t and u are **any** two tuples*)



If t and u agree here
Then they must agree here

Why we call “**functional**” dependency?

Functional Dependency (cont.)

- Some examples

Beers(name, manf)

name \rightarrow manf manf \rightarrow name ?

Sells(bar, beer, price)

Bar, beer \rightarrow price

Splitting Right Sides of FD's

- $X \rightarrow A_1 A_2 \dots A_n$ holds for R exactly when each of $X \rightarrow A_1$, $X \rightarrow A_2, \dots, X \rightarrow A_n$ hold for R .

Example: $A \rightarrow BC$ is equivalent to $A \rightarrow B$ and $A \rightarrow C$.

- There is no splitting rule for left sides.

Trivial Functional Dependencies

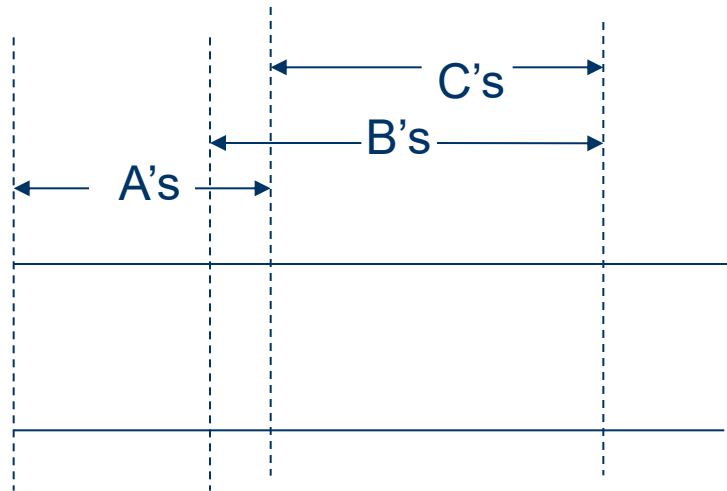
Sells(bar, beer, price)

bar, beer \rightarrow bar (trivial functional dependencies)

bar, beer \rightarrow price (nontrivial function dependencies)

A's \rightarrow B's

A's \rightarrow C's



Example: FD's

Drinkers(name, addr, beersLiked, manf, favBeer)

- Reasonable FD's to assert:
 1. name \rightarrow addr favBeer (combining rule)
 - ◆ Note this FD is the same as name \rightarrow addr and name \rightarrow favBeer. (splitting rule)
 2. beersLiked \rightarrow manf

Example: Possible Data

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

Because name -> addr

Because name -> favBeer

Because beersLiked -> manf

For example, given data \rightarrow FD's

A	B	C	D
1	2	3	4
4	3	2	1
5	1	2	3
5	1	2	9
4	3	1	2

- FD:
 $AB \rightarrow C$? \times
 $A \rightarrow B$? \checkmark

Keys of Relations

- K is a *superkey* for relation R if K functionally determines all of R .
- K is a *key* for R if K is a superkey, but no proper subset of K is a superkey.
(minimality)

Example: Superkey

Drinkers(name, addr, beersLiked, manf, favBeer)

- {name, beersLiked} is a superkey because together these attributes determine all the other attributes.
 - name -> addr favBeer
 - beersLiked -> manf
- Other superkey such as {name, beersLiked, addr}, {name,beersLiked, manf}, ...any superset of {name, beersLiked}.

Example: Key

- {name, beersLiked} is a key because neither {name} nor {beersLiked} is a superkey.
 - name doesn't -> manf;
 - beersLiked doesn't -> addr.
- Keys are also superkeys.

Where Do Keys Come From?

1. Just assert a key K .
 - The only FD's are $K \rightarrow A$ for all attributes A .
2. Assert FD's and deduce the keys by systematic exploration.
3. More FD's From "Physics"
 - Example: "no two courses can meet in the same room at the same time" tells us:
 $hour\ room \rightarrow course$.

Inferring FD's

Three ways to infer $y \rightarrow B$:

1. A simple test for it
2. Use FD to deduce
3. Calculate closure of y

- Given a relation R and its FD's $X_1 \rightarrow A_1, X_2 \rightarrow A_2, \dots, X_n \rightarrow A_n$,
→ whether an FD $Y \rightarrow B$ must hold in any relation (instance) in this R .
 - Example: If $Y \rightarrow A$ and $A \rightarrow B$ hold, we want to **infer** whether $Y \rightarrow B$ holds.

1: a simple test

- To test if $Y \rightarrow B$, start by assuming two tuples agree in all attributes of Y .

$\leftarrow Y \rightarrow$	B
0000000...0	
00000?? .. ?	

2: Use given FDs to deduce

- $R(Y,A,B)$ with FD's:
 $Y \rightarrow A, A \rightarrow B$
- To prove $Y \rightarrow B$?

Y	A	B
Y	a1	b1
Y	a2	b2

Inference steps:

- 1) Assume two tuples that agree on Y
- 2) Because $Y \rightarrow A$,
 $a1=a2$
- 3) Because $A \rightarrow B$
 $b1=b2$

Many Inference rules

- **Reflexivity:**

If $\{B_1, B_2, \dots, B_m\} \subseteq \{A_1, A_2, \dots, A_n\}$ then

$A_1, A_2, \dots, A_n \rightarrow B_1, B_2, \dots, B_m$ called trivial FD's

- **Augmentation:**

If $A_1, A_2, \dots, A_n \rightarrow B_1, B_2, \dots, B_m$ then,

$A_1, A_2, \dots, A_n, C_1, C_2, \dots, C_k \rightarrow B_1, B_2, \dots, B_m, C_1, C_2, \dots, C_k$

- **Transitivity:**

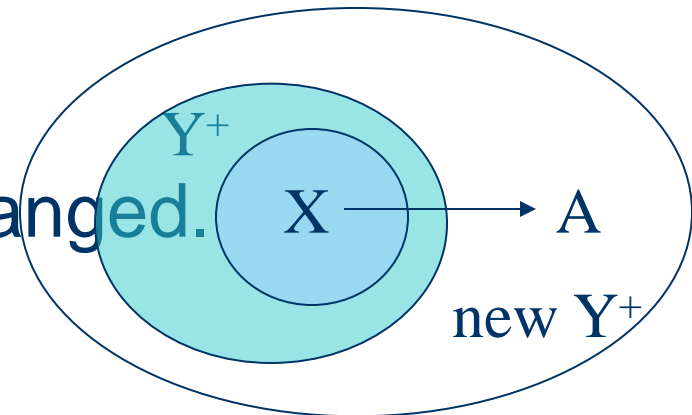
If $A_1, A_2, \dots, A_n \rightarrow B_1, B_2, \dots, B_m$, and $B_1, B_2, \dots, B_m \rightarrow C_1, C_2, \dots, C_k$

then, $A_1, A_2, \dots, A_n \rightarrow C_1, C_2, \dots, C_k$

3: Closure Test

- An easier way to test is to compute the *closure* of Y , denoted Y^+ .
- **Basis:** $Y^+ = Y$.
- **Induction:** Look for an FD's left side X that is a subset of the current Y^+ . If the FD is $X \rightarrow A$, add A to Y^+ .

- **End:** when Y^+ can not be changed.



3: Closure Test: example

- $R(Y,A,B)$ with FD's: $Y \rightarrow A, A \rightarrow B$
- To prove $Y \rightarrow B$?

- Calculating steps for Y^+ :

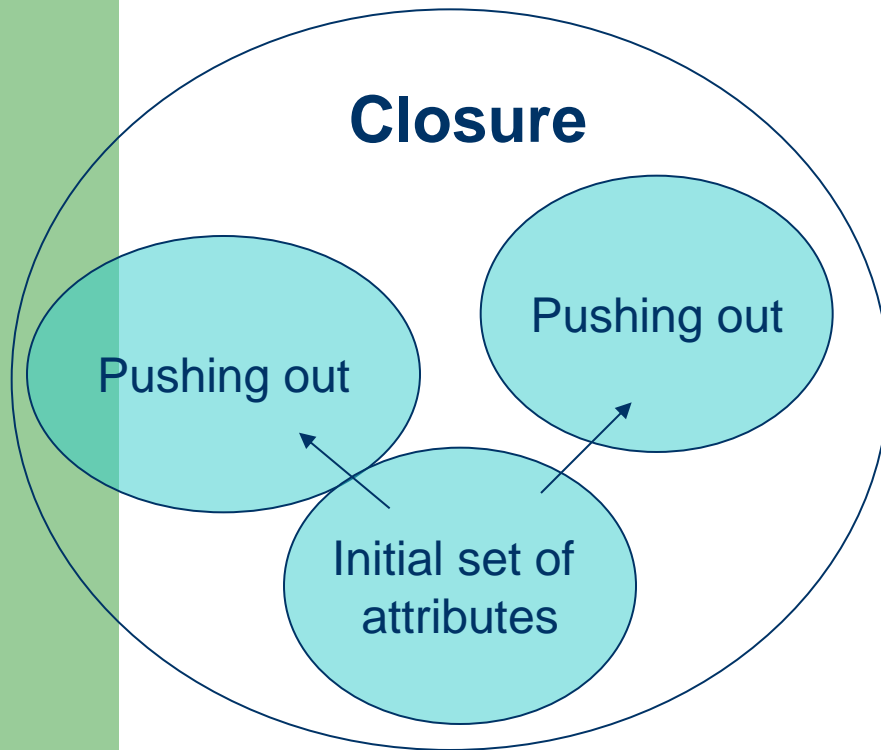
1. $Y^+ = Y$

2. $Y^+ = Y, A$

3. $Y^+ = Y, A, B \longrightarrow Y \rightarrow B$

Closure and Keys:
if the closure of X
is all attributes of a
relation, then X is
a key /superkey.

Computing the closure of a set of attributes



- The closure algorithm 3.7 (pp.76) can **discovers all true FD's**.
- We need a FD's (**minimal basis**) to represent the full set of FD's for a relation.

Basis for Functional Dependencies

- Example: $R(A,B,C)$ with all FD's:
 $A \rightarrow B, A \rightarrow C, B \rightarrow A, B \rightarrow C, C \rightarrow A, C \rightarrow B,$
 $AB \rightarrow C, AC \rightarrow B, BC \rightarrow A, \dots$
- We are free to choose any **basis** for the FD's of R , a set of FD's that can infer all the FD's that hold for R :

FD1: $A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow B$

FD2: $A \rightarrow B, B \rightarrow C, C \rightarrow A$

Given Versus Implied FD's

- **Given**: FD's that are known to hold for a relation R
- **Implied FD's** : other FD's may follow logically from the given FD's

Example:

- R(A,B,C) with FD's: $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow A$
- $A \rightarrow C$ is implied FD

Finding All Implied FD's

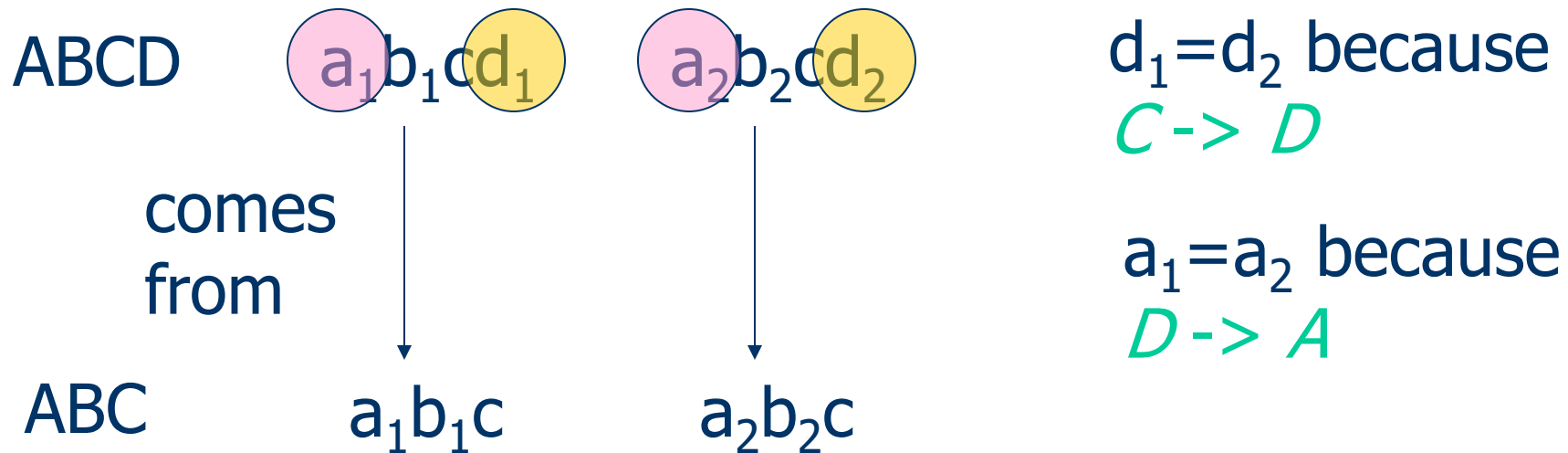
- **Motivation:** “normalization” the process where we break a relation schema into two or more schemas.
- Example: $ABCD$ with FD's $AB \rightarrow C$, $C \rightarrow D$, and $D \rightarrow A$.
 - Decompose into ABC , AD . What FD's hold in ABC ?

Not only $AB \rightarrow C$, but also $C \rightarrow A$!

Why?

$C \rightarrow A$

$ABCD$ with FD's $AB \rightarrow C$, $C \rightarrow D$,
and $D \rightarrow A$.



Thus, tuples in the projection
with equal C's have equal A's;
 $C \rightarrow A$.

Basic Idea for **projecting** functional dependencies

1. Start with given FD's and find all *nontrivial* FD's that follow from the given FD's.
 - Nontrivial = right side not contained in the left.
2. Restrict to those FD's that involve only attributes of the projected schema.

An algorithm of projecting FD's

1. For each set of attributes X , compute X^+ .
2. Add $X \rightarrow A$ for all A in $X^+ - X$.
3. However, drop $XY \rightarrow A$ whenever we discover $X \rightarrow A$.
 - ◆ Because $XY \rightarrow A$ follows from $X \rightarrow A$ in any projection.
4. Finally, use only FD's involving projected attributes.

Example: Projecting FD's

If we find $X^+ =$ all attributes, so is the closure of any **superset** of X .

- ABC with FD's $A \rightarrow B$ and $B \rightarrow C$. Project onto AC .
 - $A^+ = ABC$; yields $A \rightarrow B, A \rightarrow C$.
 - We do not need to compute AB^+ or AC^+ .
 - $B^+ = BC$; yields $B \rightarrow C$.
 - $C^+ = C$; yields nothing.
 - $BC^+ = BC$; yields nothing.
- Resulting FD's: $A \rightarrow B, A \rightarrow C$, and $B \rightarrow C$.
- Projection onto AC : $A \rightarrow C$.
 - Only FD that involves a subset of $\{A, C\}$.

Classroom Exercises

- Consider a relation $R(A,B,C,D,E)$ with the following functional dependencies:
 $A \rightarrow B$, $CD \rightarrow E$, $E \rightarrow A$, $B \rightarrow D$
- Specify all **keys** for R
- ◆ Keys: AC , BC , CD , CE

Relational Schema Design

- Goal of relational schema design is to avoid anomalies and redundancy.
 - *Update anomaly* : one occurrence of a fact is changed, but not all occurrences.
 - *Deletion anomaly* : valid fact is lost when a tuple is deleted.

Example of Bad Design

Drinkers(name, addr, beersLiked, manf, favBeer)

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

Data is redundant, because each of the ???'s can be figured out by using the FD's name -> addr favBeer and beersLiked -> manf.

This Bad Design Exhibits Anomalies

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

- **Update anomaly:** if Janeway is transferred to *Intrepid*, will we remember to change each of her tuples?
- **Deletion anomaly:** If nobody likes Bud, we lose track of the fact that Anheuser-Busch manufactures Bud.

Solve the Problem

Problems caused by FD's

Drinkers(name, addr, beersLiked, manf, favBeer) →
decompose into smaller relations :

Drinker= projection (name,addr, favBeer) (Drinkers)

Likes= projection (name, beersLiked) (Drinkers)

Beer = projection (beersliked, manf) (Drinkers)

Drinkers = Drinker **natural join** Likes **natural join** beer
not more, not less

Solve the problem (cont.)

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

decomposition



Janeway	voyager	WickedAle
Spock	Enterprise	Bud

Janeway	Bud
Janeway	WickedAle
Spock	Bud

Bud	A.B.
WickedAle	Peter's

Any anomalies?

Remember our questions:

- Why do we design relations like the example? → good design
- What makes a good relational database schema? → no redundancy, no Update/delete anomalies,
- what we can do if it has flaws? → decomposition

New Question:

- any standards for a good design?
→ Normal forms: a condition on a relation schema that will eliminate problems
- any standards or methods for a decomposition?
→ yes.