By Marina Barsky

Design Theory for Relational Databases

Lecture 15

Functional dependencies: formal definition

- $X \rightarrow Y$ is an assertion about a relation R that whenever two tuples of R agree on all the attributes 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,… represent single attributes.
- Convention: no set formers in sets of attributes, just ABC, rather than $\{A,B,C\}$.

Formal example of FDs

- AC → B

A	B	C
5	3	2
5	4	3
5	5	2

Does this instance violate AC → B?

Formal example of FDs

- AC → B

A	B	C
5	3	**2**
5	4	3
5	5	**2**

Does this instance violate AC → B?

Example: BBD

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

- name → addr
- beersliked → manf
- name → favBeer

Example: BBD

name	**addr**	**beersliked**	**manf**	**favBeer**
Janeway	Voyager	Bud	A.B.	**WickedAle**
Janeway	**Voyager**	WickedAle	Pete’s	WickedAle
Spock	Enterprise	Bud	**A.B.**	Bud

- name → addr
- beersliked → manf
- name → favBeer

Keys: formal definition

- *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

Formal example of keys

- Suppose R is a relation with attributes A, B, C

- Tell how many superkeys R has if the only key is A?

Formal example of keys

- Suppose R is a relation with attributes A, B, C
- Tell how many superkeys R has if the only key is A?
- Superkeys:
  - A
  - AB
  - ABC
  - AC

Example: BBD

name	**addr**	**favBeer**
Janeway	Voyager	Bud
Monk	Myway	WickedAle
Spock	Enterprise	Bud

- The key: *name*

Example: BBD

name	**addr**	**favBeer**
Janeway	Voyager	Bud
Monk	Myway	WickedAle
Spock	Enterprise	Bud

- The key: *name*
- Superkeys:
  {name, addr}
  {name, addr, favBeer}
  {name, favBeer}

How about {addr, favBeer}?

Inferring FD's

- We are given FD's $X_1 \rightarrow A_1$, $X_2 \rightarrow A_2$,…, $X_n \rightarrow A_n$, and we want to know whether an FD $Y \rightarrow B$ must hold in any relation that satisfies the given FD's.

- Example: If $A \rightarrow B$ and $B \rightarrow C$ hold, does $A \rightarrow C$ hold?

Inference rules

- Splitting rule
- Transitive rule
- Trivial FDs
- Closure

Splitting (and combining) rule

- Splitting right sides of FD's:
  - $X \rightarrow A_1A_2\ldots A_n$ holds for R precisely when each of $X \rightarrow A_1$, $X \rightarrow A_2,\ldots,$ $X \rightarrow A_n$ hold for R.
- Combining right sides of FD's:
  - when $X \rightarrow A_1$, $X \rightarrow A_2,\ldots,$ $X \rightarrow A_n$ hold then $X \rightarrow A_1A_2\ldots A_n$ holds
- **There is no splitting (combining) rule for left sides!**
- We'll generally express FD's with singleton right sides

Splitting rule reasoning

- Suppose we have A → BC
- This is an assertion that if 2 tuples agree on A, they also agree in all B and C
- That means that they agree in B and they agree in C: A → B, A → C

A → BC			A → B	A → C
(a, b, c)	(a, b, c)		(a, b, c)	(a, b, c)
(a, b1, c1)	(a, b, c)	⇒	(a, b, c1)	(a, b1, c)
(a, b2, c2)	(a, b, c)		(a, b, c2)	(a, b2, c)

Example: BBD

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

name -> {addr, favBeer}

- The same as:

name -> addr

name -> favBeer

{name, beersLiked} -> manf

- The same as:

name -> manf

beersLiked -> manf

Example: BBD

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

name -> {addr, favBeer}

- The same as:

name -> addr

name -> favBeer

{name, beersLiked} -> manf

- Not the same as:

~~name -> manf~~

~~beersLiked -> manf~~

Inference rules

- Splitting rule
- Transitive rule
- Trivial FDs
- Closure

Transitive rule

- If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

$A \rightarrow B$

(a, b, c)
(a, b1, c1)
(a, b2, c2)

$A \rightarrow B$

(a, b, c)
(a, b, c1)
(a, b, c2)

$B \rightarrow C$

(a, b, c)
(a, b, c1)
(a, b, c2)

$B \rightarrow C$

(a, b, c)
(a, b, c)
(a, b, c)

$A \rightarrow C$

(a, b, c)
(a, b, c)
(a, b, c)

Inference rules

- Splitting rule
- Transitive rule
- Trivial FDs
- Closure

Trivial FD's

- If $X \rightarrow Y$ and $Y \subseteq X$ then $X \rightarrow Y$ is called a trivial dependency

- Explanation: All tuples that agree in all of X surely agree in a subset of them

- Example: $AB \rightarrow B$ is a trivial dependency

Inference Test

- To test if $Y \to B$, start by assuming two tuples agree in all attributes of Y
- Use the given FD's to infer that these tuples must also agree in certain other attributes.
  - If B is one of these attributes, then $Y \to B$ is true.
  - Otherwise, the two tuples, with any forced equalities, form a two-tuple relation that proves $Y \to B$ does not follow from the given FD's.

Inference rules

- Splitting rule
- Transitive rule
- Trivial FDs
- Closure

Closure for a set of attributes Y

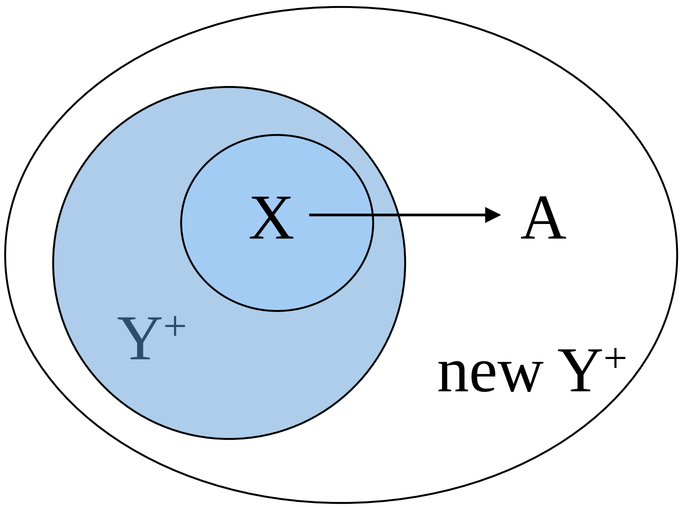
- The *closure* of a set Φ of functional dependencies is the set of all functional dependencies logically implied by Φ

- The closure for an attribute set Y is a set of all implied dependencies with Y in the left-hand side

- The *closure* of Y is denoted Y^{+}.

Computing closure for a set of attributes Y

- Convert all FDs to LHS-singleton FD's using splitting rule
- 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^+.



Example: computing closure

- Given:

R(A,B,C,D) with FD's AB → C, B → D, CD → A, AD → B.

- Computing closure for **AB**:

$\{AB\}^+ = \{ABC\}$ (from AB → C)

$\{ABC\}^+ = \{ABCD\}$ (from B → D)

- Answer:

$\{AB\}^+ = \{ABCD\}$

Example: computing closure

- Given:

R(A,B,C,D) with FD's AB → C, B → D, CD → A, AD → B.

- Computing closure for **B**:

$\{B\}^+ = \{BD\}$ (from B → D)

- Answer:

$\mathbf{\{B\}^+ = \{BD\}}$

Example: computing closure

- Given:

R(A,B,C,D) with FD’s AB → C, B → D, CD → A, AD → B.

- Computing closure for CD:

$\{CD\}^+ = \{CDA\}$ (from CD → A)

{CDA}+={CDAB} (from AD → B)

- Answer:

$\{CD\}^+ = \{ABCD\}$

Example: computing closure

- Given:

R(A,B,C,D) with FD's AB → C, B → D, CD → A, AD → B.

- Computing closure for **AD**:

$\{AD\}^+ = \{ADB\}$ (from AD → B)

$\{ADB\}+=\{ADBC\}$ (from AB → C)

- Answer:

$\{AD\}^+ = \{ABCD\}$

Why do we need to compute closure

- By computing closure for every possible set of attributes we obtain a full exhaustive set of FD’s – both declared and implied

- Closure has multiple applications

Using closure to test for an FD

- Suppose R(A,B,C,D,E,F) and the the FD's are
- AB→C, BC→AD, D→E, and CF→B

- We wish to test whether AB→D follows from the set of FD's?

- We compute $\{A,B\}^+$ which is {A,B,C,D,E}.
- Since D is a member of the closure, we imply AB→D

Using closure to test for an FD

- Consider the relation R(A, B, C, D, E) and the set of FD's S1 = {AB->C, AE->D, D->B}

- Which of the following assumptions does not follow from S1

1. S2={AD->C}
2. S2={AD->C, AE->B}
3. S2 = {ABC->D, D->B}
4. S2 = {ADE->BC}

Using closure to test for a key

One way of testing if a set of attributes, let's say A, is a key, is:

1. Find it's closure A^+.
2. Make sure that it contains all attributes of R.
3. Make sure that you cannot create a smaller set, let's say A', by removing one or more attributes from A, that has the property 2.

Using closure to compute all superkeys

- Given:

R(A,B,C,D) with FD's AB → C, B → D, CD → A, AD → B.

$\{AB\}^+ = \{ABCD\}$

$\{B\}^+ = \{BD\}$

$\{CD\}^+ = \{ABCD\}$

$\{AD\}^+ = \{ABCD\}$

{AB}, {CD}, {AD} are superkeys

Using superkeys for identifying candidate keys

R(A,B,C,D) with FD's AB → C, B → D, CD → A, AD → B.

{AB}, {CD}, {AD} are superkeys

Can A be a key?

$\{A\}^+$ = {A} – no

Can B be a key?

$\{B\}^+$ = {BD} – no

{AB} is a key – minimal superkey

Analogous tests show that {CD} and {AD} are also keys

Boyce-Codd Normal Form: formal definition

- Boyce-Codd Normal Form (BCNF): simple condition under which all the anomalies of 2NF, 3NF and BCNF can be guaranteed not to exist.

- A relation is in **BCNF** if:

  Whenever there is a *nontrivial* dependency

  $A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_m$

  for **R**, it must be the case that

  { $A_1, A_2, \ldots, A_n$ **}** is a **superkey** for **R**.

One more time: relation is in BCNF when

whenever ***X*** → ***Y*** is a nontrivial FD that holds in *R*,

X is a **superkey**

- Remember: *nontrivial* means *Y* is not contained in *X*.
- Remember, a *superkey* is any superset of a key (not necessarily a proper superset).

Example

Drinkers(<u>name</u>, addr, <u>beersLiked</u>, manf, favBeer)

- **FD's**: name → {addr, favBeer}, beersLiked->manf
- Only **key** is {name, beersLiked}.

- In each FD, the left side is *not* a superkey.
- Any one of these FD's shows *Drinkers* is **not in BCNF**

Another Example

Beers(name, manf, manfAddr)

- FD's: name → manf,   manf → manfAddr
- Only key is {name} .

- name → manf - does not violate BCNF
- manf → manfAddr - violation

Decomposition into BCNF

- Find a non-trivial FD $A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_m$ that violates BCNF, i.e. $A_1A_2 \ldots A_n$ isn't a superkey.
- Decompose the relation into two overlapping relations:
  - One is all the attributes involved in the violating dependency and
  - the other is the **left side of the violating FD** and all the other attributes not involved in the violating FD

- By repeatedly, choosing suitable decompositions, we can break any relation schema into a collection of smaller relations, each in BCNF.

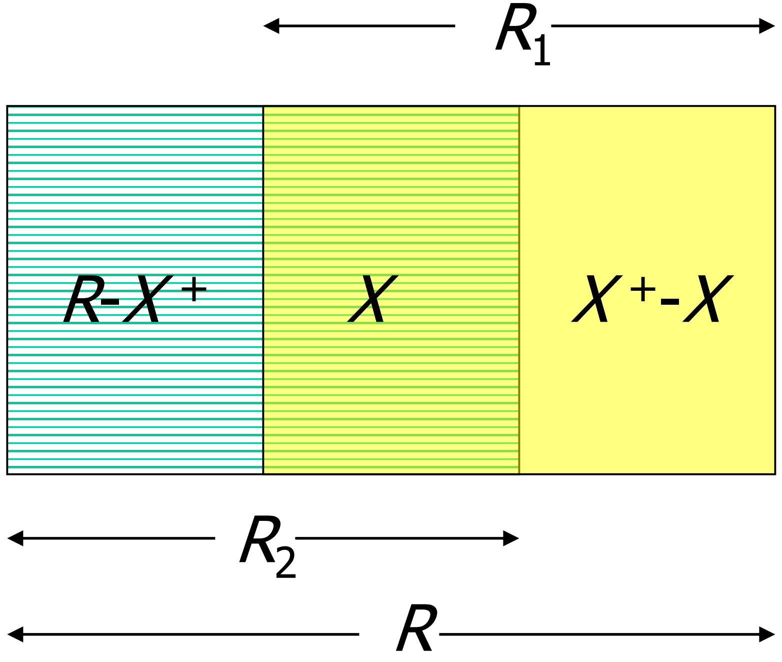
BCNF decomposition algorithm: step 1

- Given: relation *R* with FD's *F*

- Look among the given FD's for a BCNF violation $X \rightarrow Y$
- Compute X^+.
  - Not all attributes, or else X is a superkey

BCNF decomposition algorithm: step 2

- Replace R by relations with schemas:
  1. $R_1 = X^+$
  2. $R_2 = R - (X^+ - X)$



BCNF decomposition algorithm: step 3

- Identify all new FD’s in R1 and R2
- For each R1 and R2 – if any dependency violates BCNF - go to step 1

- Until no more BCNF violations

Formal Example 1/5

- Given R(A,B,C,D) with AB → C, C → D, and D → A
- Indicate all BCNF violations

$\{AB\}^+=\{ABCD\}$ - not a violation, {AB} is (super)key

$C^+ = \{CDA\}$ – violation

$D^+ = \{DA\}$ - violation

Formal Example 2/5

- Given R(A,B,C,D) with AB → C, C → D, and D → A

C^+ = {CDA} – violation

D^+ = {DA} – violation

- Decompose into relations that are in BCNF

- Variant 1:

R1 (C, D, A}

R2 (B, C)

Formal Example 3/5

- Given R(A,B,C,D) with AB → C, C → D, and D → A

C^+ = {CDA} – violation

D^+ = {DA} – violation

- Decompose into relations that are in BCNF

- Variant 2:

R1 (D, A}

R2 (B, C, D)

Formal example 4/5

- R(A,B,C,D) with AB → C, C → D, and D → A

R1 (C, D, A}

R2 (B, C)

- Should we stop? No, we need to test R1 and R2 for BCNF violations

- Which FD's do we have in R1?

C → D, and D → A

C^+ = {CDA} – not a violation

D^+ = {DA} - violation

Formal example 5/5

- R(A,B,C,D) with AB → C, C → D, and D → A

R1 (C, D, A}

R2 (B, C)

- Decomposing R1 with C → D, and D → A

D^+ = {DA} – violation

R1.1 (D,A)

R1.2 (C, D)

Final result

- R(A,B,C,D) with AB → C, C → D, and D → A

- Decomposed into:

R2 (B, C)

R1.1 (D,A)

R1.2 (C, D)

- Should we decompose any further?
- No, because every relation with 2 attributes is automatically in BCNF

Every relation with 2 attributes is in BCNF

- R (A, B)

3 cases:

- There are no non-trivial FD's

No violations

- A → B holds

A is the key – no violations

- B → A holds

B is the key no violations

■

Example: BCNF Decomposition

Drinkers(<u>name</u>, addr, <u>beersLiked</u>, manf, favBeer)

F = {name → addr, name → favBeer, beersLiked → manf}

Key: {name, addr}

- Pick BCNF violation name->addr
- Closure for the left side: $\{name\}^+$ = {name, addr, favBeer}
- Decomposed relations:

  Drinkers1(<u>name</u>, addr, favBeer)

  Drinkers2(<u>name</u>, <u>beersLiked</u>, manf)

Example: continued

- We are not done; we need to check Drinkers1 and Drinkers2 for BCNF

- For Drinkers1(name, addr, favBeer),

relevant FD's are name → addr and name → favBeer

- Thus, {name} is the only key and Drinkers1 is in BCNF

Example: continued

- For Drinkers2(name, beersLiked, manf), the only FD is beersLiked → manf, and the only key is {name, beersLiked}.
- Violation of BCNF.

beersLiked$^+$ = {beersLiked, manf}

- so we decompose *Drinkers2* into:

  Drinkers3(beersLiked, manf)

  Drinkers4(name, beersLiked)

Example: concluded

- The resulting decomposition of *Drinkers* :
  Drinkers1(name, addr, favBeer)
  Drinkers3(beersLiked, manf)
  Drinkers4(name, beersLiked)

- Notice: *Drinkers1* tells us about drinkers, *Drinkers3* tells us about beers, and *Drinkers4* tells us the relationship between drinkers and the beers they like.