Design Theory for Relational Databases

FUNCTIONAL DEPENDENCIES

Functional Dependencies

X ->*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* ->*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*}.

Functional Dependencies

3.1.1 Definition of Functional Dependency

A *functional dependency* (FD) on a relation R is a statement of the form "If two tuples of R agree on all of the attributes $A_1, A_2, \ldots, A_n$ (i.e., the tuples have the same values in their respective components for each of these attributes), then they must also agree on all of another list of attributes $B_1, B_2, \ldots, B_m$. We write this FD formally as $A_1 A_2 \cdots A_n \rightarrow B_1 B_2 \cdots B_m$ and say that

"$A_1, A_2, \ldots, A_n$ functionally determine $B_1, B_2, \ldots, B_m$"

Figure 3.1 suggests what this FD tells us about any two tuples t and u in the relation R. However, the A's and B's can be anywhere; it is not necessary for the A's and B's to appear consecutively or for the A's to precede the B's.

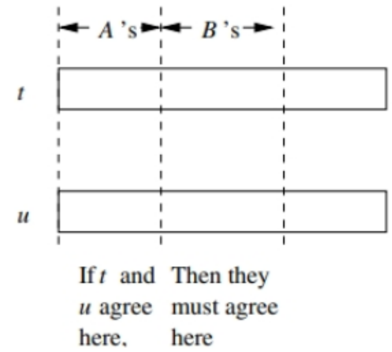


Figure 3.1: The effect of a functional dependency on two tuples.

Splitting Right Sides of FD's

$X \text{->} A_1 A_2 \ldots A_n$ holds for R exactly when each of $X \text{->} A_1$, $X \text{->} A_2, \ldots, X \text{->} A_n$ hold for R.

Example: $A \text{->} BC$ is equivalent to $A \text{->} B$ and $A \text{->} C$.

- <u>There is no splitting rule for left sides</u>.

We'll generally express FD's with singleton right sides.

Example: FD's

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

Reasonable FD's to assert:

1. name -> addr, favBeer
   - Note this FD is the same as name -> addr and name -> favBeer.
2. beersLiked -> 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

Example/Task

Define which Functional Depenendency hold over movie database?

Keys of Relations

Set of attributes K is a ***superkey*** for relation *R* if *K* **functionally determines** all attributes of *R*.

K is a ***key*** for *R* if *K* is a superkey, but **no proper subset** of *K* is a superkey.

Example: Superkey

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

Is {name, beersLiked} a superkey?

Example: Superkey

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

{name, beersLiked} is a superkey because together these attributes determine all the other attributes.

- name, beersLiked -> addr, favBeer, beersLiked, manf, favBeer

Example: Key

Is {name, beersLiked} a key?

Example: Key

{name, beersLiked} is a key because neither {name} nor {beersLiked} is a superkey.

- name doesn't -> manf; beersLiked doesn't -> addr.

There are no other keys, but lots of superkeys.

- Any superset that contains {name, beersLiked}.

Where Do Keys Come From?

1. Just assert a key *K*.
   - The only FD's are *K -> A* for all attributes *A.*
2. Assert FD's and deduce the keys by systematic exploration.

More FD’s

Example: “no two courses can meet in the same room at the same time” tells us: hour, room -> course.

Inferring FD's

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

- Example: If A -> B and B -> C hold, surely A -> C holds, even if we don't say so.

Important for design of good relation schemas.

Inference Test

To test if *Y -> B*, start by assuming two tuples agree in all attributes of *Y*.

Y

0000000. . . 0

00000?? . . . ?

Inference Test – (2)

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 -> B* is true.
- Otherwise, the two tuples, with any forced equalities, form a two-tuple relation that proves *Y -> B* does not follow from the given FD's.

Example/Task

- Assume *ABC -> DE, E -> FG, G -> H, I -> J*.
- Is it true *ABC -> G?*
- Is it true *ABC -> GH?*
- Is it true *ABC -> GHI?*
- Use inference test to provide justification to your answer.

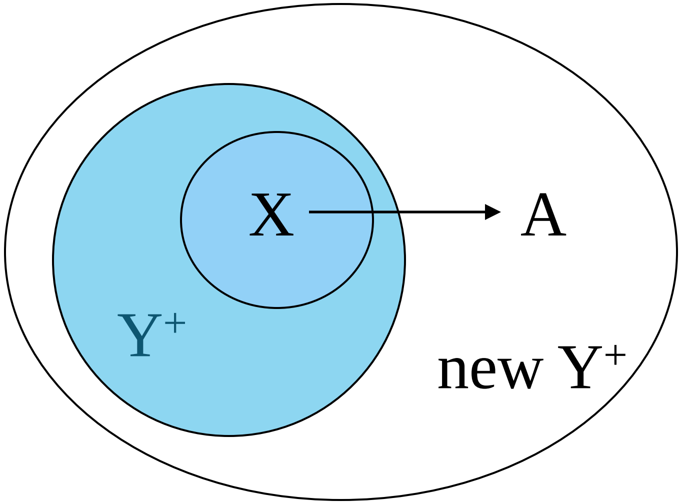
Closure Test

An easier way to test is to compute the ***closure*** of Y, denoted Y^+.

Closure Y^+ is the set of attributes that Y functionally determines.

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^+.


X
A
Y^+
new Y^+

Example – Closure Test

- Assume *ABCD -> F, ABC -> D, F -> GH, I -> JGH*.
- *What is the closure of ABC, ABC^+?*

Example – Closure Test

- Assume *ABCD -> F, ABC -> D, F -> GH, I -> JGH*.
- *What is the closure of ABC, ABC^+?*
  1. Basis $ABC^+ = ABC$

Example – Closure Test

- Assume *ABCD -> F, ABC -> D, F -> GH, I -> JGH*.
- *What is the closure of ABC, ABC^+?*
  1. Basis $ABC^+ = ABC$
  2. $ABC^+ = ABC^+$ union *D*, since ABC is a subset of ABC^+ and *ABC -> D*

Example – Closure Test

- Assume *ABCD -> F, ABC -> D, F -> GH, I -> JGH*.
- *What is the closure of ABC, ABC^+?*
  1. Basis $ABC^+ = ABC$
  2. $ABC^+ = ABC^+$ union *D*, since ABC is a subset of ABC^+ and *ABC -> D*
  3. $ABC^+ = ABC^+$ union *F*, since ABCD is a subset of ABC^+ and *ABCD -> F*

Example – Closure Test

- Assume *ABCD -> F, ABC -> D, F -> GH, I -> JGH*.
- *What is the closure of ABC, ABC^+?*

1. Basis $ABC^+ = ABC$
2. $ABC^+ = ABC^+$ union *D*, since ABC is a subset of ABC^+ and *ABC -> D*
3. $ABC^+ = ABC^+$ union *F*, since ABCD is a subset of ABC^+ and *ABCD -> F*
4. $ABC^+ = ABC^+$ plus *GH*, since *F* is a subset of ABC^+ and *F -> GH*

Therefore $ABC^+ = ABCDFGH$.

Hence, for instance, ABC -> H is true but ABC -> I is not

Task

- Assume *AB -> CH, C -> D, HD -> A, ACH -> EFG, I -> A*.
- *What is the closure of CH, CH^{+}?*

Is it true that CH -> G and CH -> B?.

Actions

Review slides!

Read Chapters 3.1. – 3.5 (Design Theory for Relational Databases).