CS54100: Database Systems

Keys and Dependencies

18 January 2012

Prof. Chris Clifton

Functional Dependencies

$X \rightarrow A$ = assertion about a relation R that whenever two tuples agree on all the attributes of X, then they must also agree on attribute A

Why do we care?

- Knowing functional dependencies provides a formal mechanism to divide up relations *(normalization)*
  - Saves space
  - Prevents storing data that violates dependencies

Fall 2007 Chris Clifton - CS541 2

Example

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

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

- Reasonable FD's to assert:

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

- Shorthand: combine FD's with common left side by concatenating their right sides.
- Sometimes, several attributes jointly determine another attribute, although neither does by itself. Example:

  beer bar → price

Keys of Relations

K is a *key* for relation *R* if:

1. $K \rightarrow$ all attributes of *R*. **(Uniqueness)**
2. For no proper subset of *K* is (1) true. **(Minimality)**

- If *K* at least satisfies (1), then *K* is a *superkey*.

Conventions

- Pick one key; underline key attributes in the relation schema.
- *X*, etc., represent sets of attributes; *A* etc., represent single attributes.
- No set formers in FD's, *e.g.*, *ABC* instead of {*A*, *B*, *C*}.

Fall 2007 Chris Clifton - CS541 5

Example

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

- {`name, beersLiked`} FD's all attributes, as seen.
  - Shows {`name, beersLiked`} is a superkey.
- `name` $\rightarrow$ `beersLiked` is false, so `name` not a superkey.
- `beersLiked` $\rightarrow$ `name` also false, so `beersLiked` not a superkey.
- Thus, {`name, beersLiked`} is a key.
- No other keys in this example.
  - Neither `name` nor `beersLiked` is on the right of any observed FD, so they must be part of *any* superkey.
- Important point: "key" in a relation refers to tuples, not the entities they represent. If an entity is represented by several tuples, then entity-key will not be the same as relation-key.

Example 2

Lastname Firstname Student ID Major

$\underbrace{\text{Lastname Firstname}}_{\text{Key (2 attributes)}}$ $\underbrace{\underbrace{\text{Student ID}}_{\text{Key}}\ \text{Major}}_{\text{Superkey}}$

Note: There are alternate keys

- Keys are {`Lastname, Firstname`} and {`StudentID`}

Who Determines Keys/FD's?

- We could assert a key *K*.
  - Then the only FD's asserted are that $K \rightarrow A$ for every attribute *A*.
  - No surprise: *K* is then the only key for those FD's, according to the formal definition of "key."
- Or, we could assert some FD's and *deduce* one or more keys by the formal definition.
  - E/R diagram implies FD's by key declarations and many-one relationship declarations.
- Rule of thumb: FD's either come from keyness, many-1 relationship, or from physics.
  - *E.g.*, "no two courses can meet in the same room at the same time" yields `room time` → `course`.

CS54100: Database Systems

Keys and Dependencies

20 January 2012

Prof. Chris Clifton

Functional Dependencies (FD's) and Many-One Relationships

- Consider $R(A1,\ldots,An)$ and X is a key then $X \rightarrow Y$ for any attributes Y in $A1,\ldots,An$ even if they overlap with X. Why?
- Suppose R is used to represent a many $\rightarrow$ one relationship:
  $E1$ entity set $\rightarrow$ $E2$ entity set
  where X key for $E1$, Y key for $E2$,
  Then, $X \rightarrow Y$ holds,
  And $Y \rightarrow X$ does not hold unless the relationship is one-one.
- What about many-many relationships?

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?"

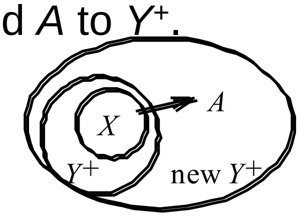
Given FD's $X1 \rightarrow A1$, $X2 \rightarrow A2$,…, $Xn \rightarrow An$, does FD $Y \rightarrow B$ necessarily hold in the same relation?

- 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^+.

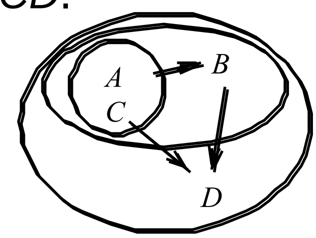


- End when Y^+ cannot be changed.

Example

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

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



Fall 2007 Chris Clifton - CS541 13

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*.

Fall 2007 Chris Clifton - CS541 14

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^+.
  - But skip $X = \emptyset$, X = all attributes.
  - Add $X \rightarrow A$ for each *A* in $X^+ - X$.
- 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*.
- Finally, project the FD's by selecting only those FD's that involve only the attributes of the projection.
  - Notice that after we project the discovered FD's onto some relation, the eliminated FD's can be inferred *in the projected relation*.

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: $C \rightarrow A$, $AB \rightarrow D$, $BD \rightarrow C$.

Example 2

- Set of FD's in *ABCGHI*:

  $A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H$

- Compute $(CG)^+$, $(BG)^+$, $(AG)^+$

Fall 2007 Chris Clifton - CS541 18

Example 3

In ABC with FD's $A \rightarrow B$, $B \rightarrow C$, project onto AC.

1. $A^+ = ABC$; yields $A \rightarrow B$, $A \rightarrow C$.
2. $B^+ = BC$; yields $B \rightarrow C$.
3. $AB^+ = ABC$; yields $AB \rightarrow C$;
   - drop in favor of $A \rightarrow C$
4. $AC^+ = ABC$ yields $AC \rightarrow B$;
   - drop in favor of $A \rightarrow B$
5. $C^+ = C$ and $BC^+ = BC$; adds nothing.

- Resulting FD's: $A \rightarrow B$, $A \rightarrow C$, $B \rightarrow C$.
- Projection onto AC: $A \rightarrow C$.

FDs: Armstrong's Axioms

- Reflexivity:
  - If $\{B_1, B_2, \ldots, B_m\} \subseteq \{A_1, A_2, \ldots, A_n\} \Rightarrow A_1A_2\cdots A_n \rightarrow B_1B_2\cdots B_m$
  - Also called "trivial FDs"
- Augmentation:
  - $A_1A_2\cdots A_n \rightarrow B_1B_2\cdots B_m \Rightarrow A_1A_2\cdots A_nC_1C_2\cdots C_k \rightarrow B_1B_2\cdots B_mC_1C_2\cdots C_k$
- Transitivity:
  - $A_1A_2\cdots A_n \rightarrow B_1B_2\cdots B_m$ and $B_1B_2\cdots B_m \rightarrow C_1C_2\cdots C_k \Rightarrow A_1A_2\cdots A_n \rightarrow C_1C_2\cdots C_k$

PURDUE UNIVERSITY

CS54100: Database Systems

Normalization

23 January 2012

Prof. Chris Clifton

Indiana Center for Database Systems ™

Normalization

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

- Formally, R is in BCNF if for every nontrivial FD for R, say $X \rightarrow A$, then X is 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.

Why are these problems?

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

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

Another Example

`Beers(name, manf, manfAddr).`

- FD's = `name` → `manf`, `manf` → `manfAddr`.
- Only key is `name`.
  - `Manf` → `manfAddr` violates BCNF with a left side unrelated to any key.

Lossless Join

- Goal: All legal values can be stored in relations
  - Recover originals through join
- Formally: X, Y is a lossless join decomposition of R w.r.t. F if $\forall r \in R$ satisfying dependencies in F, $\pi_X(r) \bowtie \pi_Y(r) = r$

CS54100

Decomposition to Reach BCNF

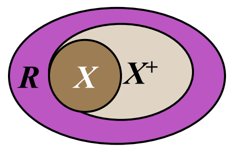
Setting: relation *R*, given FD's *F*.

Suppose relation *R* has BCNF violation $X \rightarrow B$.

- We need only look among FD's of *F* for a BCNF violation, not those that follow from *F*.
- 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.

Fall 2007 Chris Clifton - CS541 29

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

F =

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

Pick BCNF violation `name → addr`.

- Close the left side: `name` $^+$ = `name addr favoriteBeer`.
- Decomposed relations:
  `Drinkers1(name, addr, favoriteBeer)`
  `Drinkers2(name, beersLiked, manf)`
- Projected FD's (skipping a lot of work that leads nowhere interesting):
  – For `Drinkers1: name → addr` and `name → favoriteBeer`.
  – For `Drinkers2: beersLiked → manf`.

(Repeating)

- Decomposed relations:

  `Drinkers1(name, addr, favoriteBeer)`

  `Drinkers2(name, beersLiked, manf)`

- Projected FD's:
  - For `Drinkers1`: `name` → `addr` and `name` → `favoriteBeer`.
  - For `Drinkers2`: `beersLiked` → `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` → `manf` violates BCNF.

Decompose `Drinkers2`

- First set of decomposed relations:

  `Drinkers1(name, addr, favoriteBeer)`

  `Drinkers2(name, beersLiked, manf)`

- Close `beersLiked`$^+$ `= beersLiked, manf.`
- Decompose `Drinkers2` into:

  `Drinkers3(beersLiked, manf)`

  `Drinkers4(name, beersLiked)`

- Resulting relations are all in BCNF:

  `Drinkers1(name, addr, favoriteBeer)`

  `Drinkers3(beersLiked, manf)`

  `Drinkers4(name, beersLiked)`

3NF

One FD structure causes problems:

- If you decompose, you can't check all the FD's only in the decomposed relations.
- If you don't decompose, you violate BCNF.

Abstractly: $AB \rightarrow C$ and $C \rightarrow B$.

- Example 1: `title city` → `theatre` and `theatre` → `city`.
- Example 2: `street city` → `zip`, `zip` → `city`.

Keys: $\{A, B\}$ and $\{A, C\}$, but $C \rightarrow B$ has a left side that is not a superkey.

- Suggests decomposition into BC and AC.
  - But you can't check the FD $AB \rightarrow C$ in only these relations.

Example

A = `street`, B = `city`, C = `zip`.

street	zip
545 Tech Sq.	02138
545 Tech Sq.	02139

city	zip
Cambridge	02138
Cambridge	02139

`zip` → `city`

`street city` → `zip`

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 (if and only if) 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.

3NF Synthesis

- Given a canonical cover F_C for F
- Schema $S = \emptyset$
- $\forall\ A \rightarrow B \in Fc$
  - If there is no $R_i \in S$ such that $AB \subseteq R_i$
    - $S = S + AB$
- If there is no $R_i \in S$ containing a candidate key for R
  - $S = S$ + (any candidate key for R)

CS54100

PURDUE UNIVERSITY

CS54100: Database Systems

Multivalued Dependencies

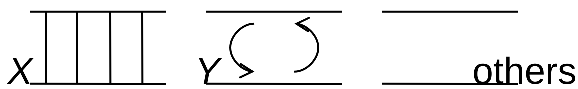
25 January 2012

Prof. Chris Clifton

Indiana Center for Database Systems ™

Multivalued Dependencies

The *multivalued dependency* $X \rightarrow\rightarrow Y$ holds in a relation R if whenever we have two tuples of R that agree in all the attributes of X, then we can swap their Y components and get two new tuples that are also in R.



Example

`Drinkers(name, addr, phones, beersLiked)` with MVD `Name` $\rightarrow\rightarrow$ `phones`. If `Drinkers` has the two tuples:

name	addr	phones	beersLiked
sue	*a*	*p*1	*b*1
sue	*a*	*p*2	*b*2

it must also have the same tuples with `phones` components swapped:

name	addr	phones	beersLiked
sue	*a*	*p*2	*b*1
sue	*a*	*p*1	*b*2

Note: we must check this condition for *all* pairs of tuples that agree on `name`, not just one pair.

MVD Rules

1. Every FD is an MVD.
   - Because if $X \rightarrow Y$, then swapping Y's between tuples that agree on X doesn't create new tuples.
   - Example, in `Drinkers: name →→ addr`.
2. *Complementation*: if $X \twoheadrightarrow Y$, then $X \twoheadrightarrow Z$, where Z is all attributes not in X or Y.
   - Example: since `name →→ phones` holds in `Drinkers`, so does `name →→ addr beersLiked`.

Splitting Doesn't Hold

Sometimes you need to have several attributes on the right of an MVD. For example:

`Drinkers(name, areaCode, phones, beersLiked, beerManf)`

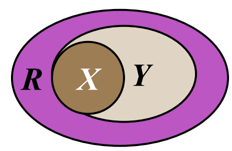
name	areaCode	phones	beersLiked	beerManf
Sue	831	555-1111	Bud	A.B.
Sue	831	555-1111	Wicked Ale	Pete's
Sue	408	555-9999	Bud	A.B.
Sue	408	555-9999	Wicked Ale	Pete's

- `name →→ areaCode phones` holds, but neither `name →→ areaCode` nor `name →→ phones` do.

4NF

Eliminate redundancy due to multiplicative effect of MVD's.

- Roughly: treat MVD's as FD's for decomposition, but not for finding keys.
- Formally: R is in Fourth Normal Form if whenever MVD $X \to\to Y$ is *nontrivial* (Y is not a subset of X, and $X \cup Y$ is not all attributes), then X is a superkey.
  - Remember, $X \to Y$ implies $X \to\to Y$, so 4NF is more stringent than BCNF.
- Decompose R, using 4NF violation $X \to\to Y$, into XY and $X \cup (R—Y)$.



Example

`Drinkers(name, addr, phones, beersLiked)`

- FD: `name` $\to$ `addr`
- Nontrivial MVD's: `name` $\to\to$ `phones` and `name` $\to\to$ `beersLiked`.
- Only key: {`name, phones, beersLiked`}
- All three dependencies above violate 4NF.
- Successive decomposition yields 4NF relations:

  D1(<u>name</u>, addr)

  D2(<u>name</u>, <u>phones</u>)

  D3(<u>name</u>, <u>beersLiked</u>)

4NF Decomposition

- Schema $S = R, D+$ be the closure of the functional and multivalued dependencies
- While $\exists\, R_i \in S$ not in 4NF w.r.t. $D+$
  - Choose a nontrivial multivalued dependency $A \rightarrow\rightarrow B$ that holds on R_i, where $A \rightarrow R_i \notin D+$, and $A \cap B = \emptyset$
  - $S = (S - R_i) \cup (R_i - B) \cup (A, B)$

CS54100