Intro to DB

CHAPTER 8
RELATIONAL DB DESIGN

Chapter 8: Relational Database Design

- Features of Good Relational Design
- Atomic Domains and First Normal Form
- Decomposition Using Functional Dependencies
- Functional Dependency Theory
- Algorithms
- Decomposition Using Multivalued Dependencies
- More Normal Form
- Database-Design Process
- Modeling Temporal Data

Pitfalls of Relational Database Design

- Relational database design
  - R = (A B C D E)   <----- single relation schema
  - $DB_1 = \{ R_1, \ldots\ldots, R_n \}$ <----- DB schema (set of relation schemas)
- Design Goals:
  - Ensure that relationships among attributes are represented (information content)
  - Avoid redundant data
  - Facilitate enforcement of database integrity constraints
- A bad design may lead to
  - Inability to represent certain information
  - Repetition of Information
  - Loss of information

Example

Lending-schema = (*branch-name, branch-city, assets, customer-name, loan-number, amount*)

branch-name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500

- Redundancy:
  -
  - Wastes space
  - Complicates updating, introducing possibility of inconsistency of *assets* value
- Null values
  - Can use null values, but they are difficult to handle.

Redundancy creates problems

- Anomalies (by Codd)
  - *Insertion anomaly*: cannot store information about a branch if no loans exist
  - *Deletion anomaly*: lose branch info when that last account for the branch is deleted
  - *Update anomaly*: what happens when you modify asset for a branch in only a single record?
-
- Solution

  decompose schema so that each information content is represented only once (later)
  - information content: relationship between attributes

First Normal Form

- Domain is atomic if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - set of names, composite attributes
    - identification numbers like CS101 that can be broken up into parts
- A relational schema is in ***first normal form (1NF)***
- Atomicity is actually a property of how the elements of the domain are used
  - Student ID numbers: *CS0012, EE1127, …*
- Non-atomic attributes leads to
  - encoding of information in the application program …
    - … rather than in the database
  - complication in storage and query processing
- We assume all relations are in first normal form

Relational Theory

Goal: Devise a theory for the following

- Decide whether a particular relation *R* is in "good" form.
- In the case that a relation *R* is not in "good" form,
  - each relation is in good form
  - the decomposition is lossless (preserves the information in the original relation before decomposition)
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies (not covered in this semester)

Functional Dependencies

- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
-

- Example
  - *Which attribute's values depend on other attributes?*

Student=(ID, Name, Dept, Dept_office, College, Dean, Advisor, Adv_phone)

Supplies=(Supplier, S-contact, Part-ID, Part-Name, Size, Proj-ID, Location, Manager, P-contact, Quantity)

Functional Dependencies (Cont.)

- Let R be a relation schema

  $$\alpha \subseteq R \ \ \textit{and} \ \ \beta \subseteq R$$

- The *functional dependency* $\alpha \rightarrow \beta$ holds on R if and only if
  - for any *legal* relations r(R),
  - whenever any two tuples t_1 and t_2 of r agree on the attributes α,
  - they also agree on the attributes β.
  - That is,

  $$t_1[\alpha] = t_2[\alpha] \ \ \Rightarrow \ \ t_1[\beta] = t_2[\beta]$$

- Example
  - Consider $r(A,B)$ with the following instance of r

1	4
1	5
3	7

  -

Applications of FD

- K is a *superkey* for relation schema R if and only if $K \rightarrow R$
- K is a *candidate key* for R if and only if
  - $K \rightarrow R$, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$
- Functional dependencies allow us t ed using superkeys.

Loan-info-schema = (*customer-name, loan-number, branch-name, amount*)

We expect the following functional dependencies to hold:

$$loan\text{-}number \rightarrow amount$$
$$loan\text{-}number \rightarrow branch\text{-}name$$

but would not expect the following to hold:

Applications of FD (Cont.)

- Specify constraints on the set of legal relations
  - We say that *F* holds on *R* if
    all legal relations on *R* satisfy the set of functional dependencies *F*
- Test relations to see if they are legal under a given set of FDs
  - If a relation *r* is legal under a set *F* of functional dependencies,
    we say that *r* satisfies *F*.
- Note:

What causes redundancy?

Lending-schema = (*b-name, b-city, assets, c-name, loan#, amount*)

branch-name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500

F = { *b-name* $\rightarrow$ *b-city assets* ; *loan#* $\rightarrow$ *amount b-name* }, Key = {*c-name, loan#*}

- Redund
  - *b-city*
  - *amount, b-name* are repeated for each loan
- Observations
  - Same values repeated for attributes that are functionally dependent on non-key attributes!

Boyce-Codd Normal Form - informally

- A relation *R* is in “good” form IF attributes are only dependent on keys
  - No non-key FDs!
  - Solution: Break *R* into smaller relations that hold tightly related attributes!

- Example

  Lending-schema = (*b-name, b-city, assets, c-name, loan#, amount*)

  F = { *b-name* $\rightarrow$ *b-city assets* ; *loan#* $\rightarrow$ *amount b-name* },   Key = {*c-name, loan#*}

=> Decompose

Branch = (*b-name, b-city, assets*)   { *b-name* $\rightarrow$ *b-city assets* }

Loan = (*loan#, amount, b-name*)   { *loan#* $\rightarrow$ *amount, b-name* }

CustLoan = (*c-name, loan#*)

Trivial FD

- *A* functional dependency is *trivial* if it is satisfied by all instances of a relation
  - *E.g.*
    - *customer-name, loan-number* $\rightarrow$ *customer-name*
    - *customer-name* $\rightarrow$ *customer-name*

- Lemma:

Closure of a Set of FDs

- Given a set *F* of FDs, there are other FDs that are *logically implied* by *F*
  - E.g. If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- **Definition**: The set of all functional dependencies logically implied by *F* is the *closure* of *F* (denoted F^+).
- We can find all of F^+ by applying *Armstrong's Axioms*:
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(*reflexivity*)**
  - if $\alpha \rightarrow \beta$, then $\gamma\, \alpha \rightarrow \gamma\, \beta$ **(*augmentation*)**
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(*transitivity*)**
- These rules are
  - sound

  and
  - complete

Example

- $R = (A, B, C, G, H, I)$ $F = \{$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H$ $\}$
- some members of F^+
  - $A \rightarrow H$
    - by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$
  - $AG \rightarrow I$
    -
  - $CG \rightarrow HI$
    - from $CG \rightarrow H$ *and* $CG \rightarrow I$: "union rule"
      - can be inferred from definition of functional dependencies, or
      - Augmentation of $CG \rightarrow I$ to infer $CG \rightarrow CGI$, augmentation of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then transitivity

Boyce-Codd Normal Form – formally

- We want a way to decide whether a particular relation R is in "good" form.
- **Definition**: A relation schema R is in *BCNF* (with respect to a set F of FDs) if for each FD $\alpha \rightarrow \beta$ in F^+ ($\alpha \subseteq R$ and $\beta \subseteq R$), at least one of the following holds:
  -
  -
- Example

  $R = (A, B, C)$, $F = \{A \rightarrow B\ ;\ B \rightarrow C\}$, Key = $\{A\}$

  - R is not in BCNF
  - Decompose into $R_1 = (A, B)$, $R_2 = (B, C)$
    - R_1 and R_2 are in BCNF
- Is the decomposed set of schemas equivalent to the original schema?

Decomposition

- Decompose schema so that each information content is represented only once
- **Definition**: Let R be a relation scheme

  $\{R_1, ..., R_n\}$ is a *decomposition* of R

  if $R = R_1 \cup ... \cup R_n$

- We will deal mostly with binary decomposition:
  - R into $\{R_1, R_2\}$ where $R = R_1 \cup R_2$

student(*ID, name, dept, dept_chair, dept_phone, year*)

=> *student'*(*ID, name, year, dept*)

department(*dept, chair, phone*)

Lending = (*b_name, asset, b_city, loan#, c_name, amount*)

=> *Branch* = (*b_name, asset, b_city*)

Loan = (*loan#, c_name, amount*)

Lossy Decomposition

- Careless decomposition leads to loss of information: *Lossy decomposition*
- Decompose schema so that each information content is represented only once

Lending = (*b_name, asset, b_city, loan#, c_name, amount*)

=> *Branch* = (*b_name, asset, b_city*)
Loan = (*loan#, c_name, amount*)

=> *Branch* = (*b_name, asset, b_city*)
Loan = (*loan#, c_name, amount, b_city*)

Lossy Decomposition (cont.)

- Decomposition of $R = (A, B)$ into

  $R_1 = (A)$ and $R_2 = (B)$

A	B
α	1
α	2
β	1

A
α
β

$\Pi_A(r)$

B
1
2

$\Pi_B(r)$

-

$$\Pi_A(r) \bowtie \Pi_B(r)$$

A	B
α	1
α	2
β	1
β	2

Lossy!

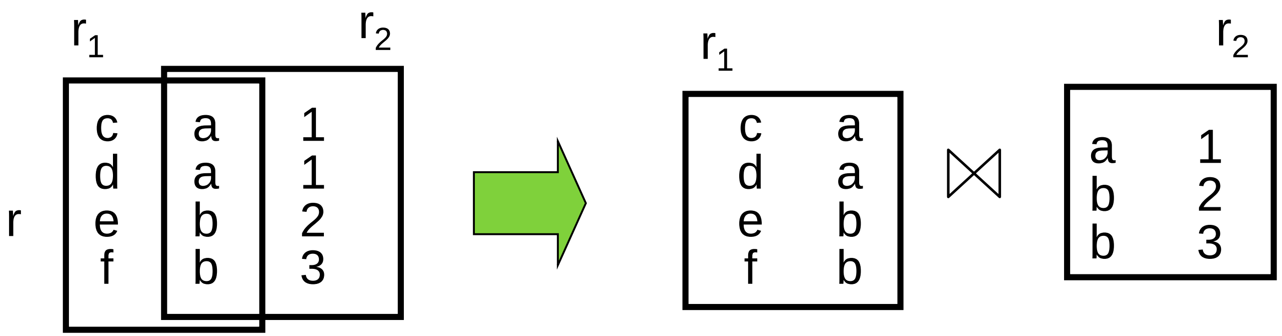
Lossless-join Decomposition

- For $r(R)$ and decomposition $\{R_1, R_2\}$, it is always the case that

- **Definition**: Decomposition $\{R_1, R_2\}$ is a *lossless-join decomposition* of R if

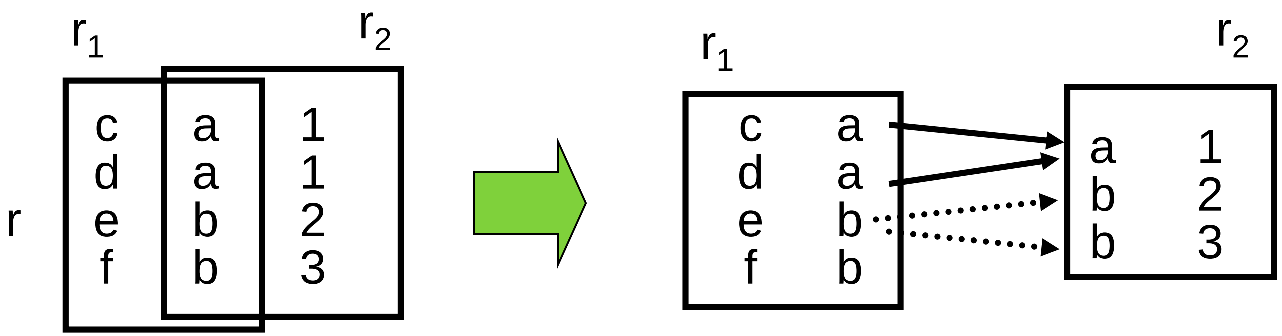
$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

  - The information content of the original relation r is always the basis



Lossless-join Decomposition

- **Lemma**: $\{R_1, R_2\}$ is a *lossless join decomposition* if
  - i.e., if one of the two sub-schemas hold the key of the other sub-schema



BCNF Example

- R = (*bname, bcity, assets, cname, loan#, amount*)

  $F = \{\ bname \rightarrow assets\ bcity\ ;\quad loan\# \rightarrow amount\ bname\ \}$

  Key = {*loan#, cname*}

Decomposition

R_1 = (*bname, bcity, assets*)

R_2 =

Final decomposition result: $\{\ R_1, R_3, R_4\ \}$

Dependency Preservation

Example

student(*name*, *dept*, *college*)	*name* $\rightarrow$ *dept*, *college*
	dept $\rightarrow$ *college*

- Decomposition 1

student1(*name*, *dept*)	*name* $\rightarrow$ *dept*
department(*dept*, *college*)	*dept* $\rightarrow$ *college*

- Decomposition 2

student1(*name*, *dept*)	*name* $\rightarrow$ *dept*
student2(*name*, *college*)	*name* $\rightarrow$ *college*

Dependency Preservation (cont.)

- **Definition**

  Let F: set of FD on R. $\{R_1, ..., R_n\}$: decomposition of R.

  The ***restriction*** of F to R_i, denoted F_i, is the set of all FDs in F^+ that include only attributes of R_i

- **Definition**

  Let $F' = F_1 \cup \ldots \cup F_n$.

  The decomposition is ***dependency-preserving*** if $F^+ = F'^+$

- *Motivation*:
  - Accessing multiple tables can be expensive
  - SQL does not provide a direct way of specifying functional dependencies other than superkeys
  - (Assertions can be ad hoc and expensive)

Example

- $R = (A, B, C)$
  $F = \{ A \rightarrow B, \ B \rightarrow C \}$

- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless-join decomposition: $R_1 \cap R_2 = \{ B \}$ and $B \rightarrow BC$
  - Dependency preserving

- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless-join decomposition: $R_1 \cap R_2 = \{ A \}$ and $A \rightarrow AB$
  -

BCNF and Dependency Preservation

St	*Zp*	*C*
s_1	z_1	c_1
s_2	z_1	c_1
s_3	z_2	c_1
null	z_3	c_2

- *R* = (*Street, City, Zip*)
  F = { *Street City* $\rightarrow$ *Zip; Zip* $\rightarrow$ *City* }
  Two candidate keys: *Street City* and *Street Zip*
  - *R* is not in BCNF
  - Any decomposition of *R* will fail to preserve
    Street City $\rightarrow$ *Zip*
-
- There are some situations where
  - BCNF is not dependency preserving, and
  - efficient checking for FD violation on updates is important

  => *solution:* define a weaker normal form

BCNF and Dependency Preservation (cont.)

- BCNF decomposition has

  R_1(*Street, Zip*)

  R_2(*Zip, City*)

- R_1, R_2 are in BCNF
  - but not dependency-preserving

    => Testing for *Street City* $\rightarrow$ *Zip* requires a join

St	*Zp*	*C*
s_1	z_1	c_1
s_2	z_1	c_1
s_3	z_2	c_1
null	z_3	c_2

St	*Zp*
s_1	z_1
s_2	z_1
s_3	z_2

Zp	*C*
z_1	c_1
z_2	c_1
z_3	c_2

Third Normal Form

- Third Normal Form
  - Allows some redundancy (with resultant problems)
  - But FDs can be checked on individual relations without a join
  - There is always a lossless-join, dependency-preserving decomposition into 3NF
- A relation schema *R* is in ***third normal form (3NF)*** if for all $\alpha \rightarrow \beta$ in F^+ at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
  - α is a superkey for *R*
  - Each attribute *A* in $\beta - \alpha$ is contained in a candidate key for *R.*

    (*NOTE:* each attribute may be in a different candidate key)
-

Example

St	Zp	C
s_1	z_1	c_1
s_2	z_1	c_1
s_3	z_2	c_1
null	z_3	c_2

R = (*Street, City, Zip*)
F = { *Street City* $\rightarrow$ *Zip*; *Zip* $\rightarrow$ *City* }

- Two candidate keys: *Street City* and *Street Zip*
- R is in 3NF
  - *Street City* $\rightarrow$ *Zip* : *Street City* is a superkey
  - *Zip* $\rightarrow$ *City* : *City* is contained in a candidate key
- But not in BCNF (nontrivial & *zip* is not key)
- There is some redundancy in this schema
  - repetition of information (e.g., the relationship z_1, c_1)
  - need to use null values (e.g., to represent the relationship z_3, c_2 where there is no corresponding value for *St*)

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into relations in 3NF and
  - the decomposition is lossless
  - the dependencies are preserved

- It is always possible to decompose a relation into relations in BCNF and
  - the decomposition is lossless
  -

Design Goals

- When we decompose a relation schema R with a set of functional dependencies F into R_1, R_2,.., R_n we want
  1. Lossless decomposition
  2. No redundancy
  3. Dependency preservation

- First, try to achieve
  - BCNF
  - Lossless join
  - Dependency preservation

- If we cannot achieve this, we accept one of
  -
  -

Algorithms

- Testing for BCNF
- BCNF Decomposition
- Testing for 3NF
- 3NF Decomposition

- Closure of FDs
- Closure of attributes
- Cover
- Canonical cover

Closure of Attribute Sets

- **Definition**: Given a set of attributes α, the *closure* of α under F (denoted by α^+) is the set of attributes that are functionally determined by α under *F:*

- Algorithm to compute α^+

  result := α;
  while (changes to *result*) **do**
      for each $\beta \rightarrow \gamma$ **in** *F* **do**
        begin
          if $\beta \subseteq$ *result* **then** *result* := *result* $\cup\ \gamma$
        end

Example

- $R = (A, B, C, G, H, I)$
- $F = \{ A \rightarrow B;\ A \rightarrow C;\quad CG \rightarrow H;\quad CG \rightarrow I;\quad B \rightarrow H \}$
- $(AG)^+$
  1. *result* =
  2. *result* =
  3. *result* =
  4. *result* =
- Is AG a candidate key?
  - Is AG a super key?
    - Does $AG \rightarrow R$?
  - Is any subset of AG a superkey?
    - Does $A^+ \rightarrow R$?
    - Does $G^+ \rightarrow R$?

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- **Testing for superkey**: “is α a superkey?”
  -

- **Testing functional dependencies**: “does $\alpha \rightarrow \beta$ hold?”
  - Or, in other words, is $\alpha \rightarrow \beta$ in F^+
  - Just check if $\beta \subseteq \alpha^+$.
  - Is a very useful simple test

- **Computing the closure of *F*:** F^+
  - For each $\gamma \subseteq R$, we find the closure γ^+, and
  - for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$.

Testing for BCNF

- **Check if $\alpha \rightarrow \beta$ cause a violation of BCNF**
  1. compute α^+ (the attribute closure of α), and
  2. verify that it includes all attributes of *R* (i.e., it is a superkey of *R*)

- **Check if *R* is in BCNF** (w.r.t. *F*)

  - It can be shown that if none of the dependencies in *F* causes a violation of BCNF, then none of the dependencies in F^+ will cause a violation of BCNF either

Testing for BCNF (cont.)

- However, using only F is *incorrect*

  when testing a relation in a *decomposition of R*

- Example

  Consider R (A, B, C, D) with $F = \{ A \rightarrow B,\ B \rightarrow C \}$

  - Decompose into $R_1(A, B)$ and $R_2(A, C, D)$
  - Neither of the dependencies in F contain only attributes from (A, C, D) so we might be mislead into thinking R_2 satisfies BCNF.
  - In fact, dependency $A \rightarrow C$ in F^+ shows R_2 is not in BCNF.

BCNF Decomposition Algorithm

result := {*R*}; *done* := false

compute F^+

while (not *done*) **do**

if (there is a schema R_i in *result* that is not in BCNF)

then begin

let $\alpha \rightarrow \beta$ $(\alpha \cap \beta = \emptyset)$ be a nontrivial FD that holds on R_i, and $\alpha \rightarrow R_i$ is not in F^+

result := (*result* – R_i) $\cup$ (R_i – β) $\cup$ (α, β);

end

else *done* := **true;**

* Each R_i in *result* is in BCNF, and decomposition is lossless-join.

R = (*bname, bcity, assets, cname, loan#, amount*)

F = { *bname*$\rightarrow$*assets bcity;*

loan# $\rightarrow$ *amount bname* }

Key={*loan#, cname*}

Decomposition

R_1 = (*bname, bcity, assets*),

R_2 = (*bname, cname, loan#, amount*)

R_3 = (*bname, loan#, amount*)

R_4 = (*cname, loan#*)

Final decomposition result:

R_1, R_3, R_4

Overall Database Design Process

- We have assumed schema *R* is given

R could have been

- a single relation containing *all* attributes that are of interest
  - called ***universal relation***
  - Normalization breaks *R* into smaller relations.
-
- the result of some ad hoc design of relations, which we then test/convert to normal form.

Denormalization for Performance

- May want to use non-normalized schema for performance
  - E.g. displaying *customer-name* along with *account-number* and *balance* requires join of *account* with *depositor*

- Alternative 1: Use ***denormalized*** relation containing attributes of *account* as well as *depositor* with all above attributes
  - faster lookup
  - extra space and extra execution time for updates
  - extra coding work for programmer and possibility of error in extra code

- Alternative 2: use a ***materialized view*** defined as

  $account \bowtie depositor$
  - benefits and drawbacks same as above
  -

Other Design Issues

- Some aspects of database design are not caught by normalization

Instead of *earnings*(*company-id, year, amount*), use

- *earnings-2000, earnings-2001, earnings-2002*, …
  - all on the schema (*company-id, earnings*).
  - above are in BCNF
  - but make querying across years difficult and
  - needs new table each year

- *company-year*(*comp-id, earnings2000, earnings2001, earnings2002*)
  - Also in BCNF
  - makes querying across years difficult and
  - requires new attribute each year.
  - Is an example of a ***crosstab***, where values for one attribute become column names => used in spreadsheets and data analysis tools

Testing for 3NF

- Need to check only FDs in F (not F^+)
- For each dependency $\alpha \rightarrow \beta$,
  - Check if α is a superkey (attribute closure check)
- If α is not a superkey
  - we have to verify if each attribute in β is contained in a candidate key
  - this test is rather more expensive, since it involves finding candidate keys
- Testing for
- Interestingly, decomposition into third normal form can be done in polynomial time

Cover

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
  - $A \rightarrow C$ is redundant in: $\{ A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow C \}$
  - Parts of a functional dependency may be redundant
    - E.g. on RHS: $\{ A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow CD \}$ can be simplified to $\{ A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D \}$
    - E.g. on LHS: $\{ A \rightarrow B,\ \ B \rightarrow C,\ \ AC \rightarrow D \}$ can be simplified to $\{ A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D \}$

-

- A FD $g \in F$ is *redundant* if $(F - \{g\})^+ = F^+$ or $g \in (F - \{g\})^+$

- F' is a *nonredundant (minimal) cover* of F if
  - $F'^+ = F^+$ and
  - F' contains no redundant FD

Extraneous Attributes

- Let $\alpha \rightarrow \beta$ in F.
  - $A \in \alpha$ is extraneous if $F \Rightarrow (F - \{\alpha \rightarrow \beta\}) \cup \{ (\alpha - A) \rightarrow \beta \}$
  - $A \in \beta$ is extraneous if $F \Leftarrow (F - \{\alpha \rightarrow \beta\}) \cup \{ \alpha \rightarrow (\beta - A) \}$
- *Note:* implication in the opposite direction is trivial, since a "stronger" functional dependency always implies a weaker one

- Example
  - Given $F = \{A \rightarrow C, AB \rightarrow C \}$
  - B is extraneous in $AB \rightarrow C$ because
    -

- Example
  - Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  - C is extraneous in $AB \rightarrow CD$ since
    -

Testing if an Attribute is Extraneous

$\alpha \rightarrow \beta \in F$

- To test if attribute $A \in \alpha$ is extraneous in α
  1. compute $(\{\alpha\} - A)^+$ using the dependencies in F
  2. A is extraneous if $(\{\alpha\} - A)^+$ contains β

- To test if attribute $B \in \beta$ is extraneous in β
  1. compute α^+ using only the dependencies in
  2. B is extraneous if α^+ contains B,

Canonical Cover

- **Definition**:

  A *canonical cover* for F is a set of dependencies F_c such that
  - $F_c^+ = F^+$
  - No FD in F_c contains an extraneous attribute
  - Each left side of a FD in F_c is unique

- Intuitively, F_c is
  - a “minimal” set equivalent to F
  -
  -

Algorithm for Canonical Cover

repeat

replace any $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_1$

with $\alpha_1 \rightarrow \beta_1 \beta_2$ (union rule)

Find $\alpha \rightarrow \beta$ with extraneous attribute either in α or β and delete the extraneous attribute from $\alpha \rightarrow \beta$

until F does not change

- Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied
- $O(n^2)$

$R = (A, B, C)$

$F = \{ A \rightarrow BC$
$B \rightarrow C$
$A \rightarrow B$
$AB \rightarrow C \}$

- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
- A is extraneous in $AB \rightarrow C$
  - $B \rightarrow C$ logically implies $AB \rightarrow C$.
- C is extraneous in $A \rightarrow BC$
  - $A \rightarrow BC$ is logically implied by $A \rightarrow B$ and $B \rightarrow C$.
- The canonical cover:

3NF Decomposition Algorithm

F_c (canonical cover for F)

$i := 0$

for each FD $\alpha \rightarrow \beta$ in F_c **do**

if no R_j, $1 \leq j \leq i$ contains $\alpha\ \beta$

then { $i := i + 1$

$R_i := \alpha\ \beta$ }

if no R_j, $1 \leq j \leq i$ contains a candidate key for R **then**

{ $i := i + 1$;

R_i := any candidate key for R }

return $(R_1, R_2, ..., R_i)$

Banker = (*branch, cname, banker, office#*)

F={ *banker* $\rightarrow$ *branch office#*

cname branch $\rightarrow$ *banker* }

Key= {*cname, branch*}

Follow the algorithm

B1 =

B2 =

Since *B2* contains a candidate key, we are done.

END OF CHAPTER 8