Relational
base
D n

10 Relational Database Design

Anomalies can be removed from relation designs by decomposing them until they are in a normal form.

Several problems should be investigated regarding a decomposition.

A decomposition of a relation scheme, R, is a set of relation schemes $\{R_1, . . . ,R_n\}$ such that $R_i \subseteq R$ for each i, and $\bigcup_{i=1}^{n} R_i = R$

Note that in a decomp of each pair of R_i and R_j does not have to be em

Example: R = {A, B, C, D, E}, R_1 = {A, B $_3$ = {C, D, E}

A naive decomposition: each relation has only attribute.

A good decomposition should have the following two properties.

Dependency Preserving

Definition: Two sets F and G of FD's are equivalent if $F^+ = G^+$.

Given a decomposition $\{R_1, \ldots, R_n\}$ of R:

$$F_i = \{X \to Y : X \to Y \in F \,\&\, X \in R_i, Y \in R_i\}.$$

The decomposition $\{R_1, \ldots, R_n\}$ of R is dependency preserving with respect to F if

$$F^+ = \left(\bigcup_{i=1}^{i=n} F_i\right)^+.$$

Examples

$F = \{ A \rightarrow BC, D \rightarrow EG, M \rightarrow A \}$, $R = (A, B, C, D, E, G, M, A)$

1) Given $R_1 = (A, B, C, M)$ and $R_2 = (C, D, E, G)$,

$F_1 = \{ A \rightarrow BC, M \rightarrow A\}$, $F_2 = \{D \rightarrow EG\}$

$F = F_1 \cup F_2$. thus, dependency preserving

2) Suppose that $F' = F \cup \{M \rightarrow D\}$. R_1 and R_2 remain the same.

Thus, F_1 and F_2 remain

We need to verify if M

Since $M^+ |_{F1 \cup F2} = \{M, A, B, C\}$, $M \rightarrow D$ is no F_2.

Thus, R_1 and R_2 are not dependency preservin

3) $F'' = \{A \rightarrow BC, D \rightarrow EG, M \rightarrow A, M \rightarrow C, C \rightarrow D, M \rightarrow D\}$

$F_1 = \{A \rightarrow BC, M \rightarrow A, M \rightarrow C\}$, $F_2 = \{D \rightarrow EG, C \rightarrow D\}$

It can be verified that $M \rightarrow D$ is inferred by F_1 and F_2.

Thus, $F''^+ = (F_1 \cup F_2)^+$

Hence, R_1 and R_2 are dependency preserving regarding F''.

Lossless Join Decomposition

A second necessary property for decomposition:

A decomposition $\{R_1, \ldots, R_n\}$ of R is a *lossless join* decomposition with respect to a set F of FD's if for every relation instance r that satisfies F:

$$(\;\;) \qquad R_n(r).$$

If $r \subset \pi R_1(r) \bowtie \cdots \bowtie \pi R_n(r)$, the decom y.

Lossless Join Decomposition(cont)

Example 2:

Suppose that we decompose the following relation:

STUDENT_ADVISOR

Name	Department	Advisor
Jones	Comp Sci	Smith
[illegible]	[illegible]	rner
[illegible]	[illegible]	sky
Dulles	Decision	[illegible]
Duke	Mathema	[illegible]
James	Comp Sci	Clark
Evan	Comp Sci	Smith
Baxter	English	Bronte

With dependencies $\{Name \rightarrow Department, Name \rightarrow Advisor, Advisor \rightarrow Department\}$, into two relations:

Lossless Join Decomposition(cont)

STUDENT_DEPARTMENT

Name	Department
Jones	Comp Sci
Ng	Chemistry
Martin	
Duke	
Dulles	Decision Sci
James	Comp Sci
Evan	Comp Sci
Baxter	English

DEPARTMENT_ADVISOR

Department	Advisor
Comp Sci	Smith
Chemistry	Turner
s	Bosky
Sci	Hall
s	James
	Clark
English	Bronte

If we join these decomposed relations we get:

Lossless Join Decomposition(cont)

Name	Department	Advisor
Jones	Comp Sci	Smith
Jones	Comp Sci	Clark*
Ng	Chemistry	Turner
Martin	Physics	Bosky
Dulles	Decision Sci	Hall
Duke	Mathematics	James
Jam	[illegible]	ith*
Jam	[illegible]	lark
Evan	Comp Sci	[illegible]
Evan	Comp Sci	*
Baxter	English	Bronte

This is not the same as the original relation (the tuples marked with * have been added). Thus the decomposition is lossy.

Useful theorem: The decomposition $\{R_1, R_2\}$ of R is lossless iff the common attributes $R_1 \cap R_2$ form a superkey for either R_1 or R_2.

Lossless Join Decomposition(cont)

Example 3: Given R(A,B,C) and F = {$A \rightarrow B$}. The decomposition into $R_1(A,B)$ and $R_2(A,C)$ is lossless because $A \rightarrow B$ is an FD over R_1, so the common attribute A is a key of R_1.

Testing for the lossless join property

Algorithm TEST_LJ

Step 1: Create a matrix S, each element $s_{i,j} \in S$ corresponds the relation R_i and the attribute A_j, such that:

$s_{j,i}$ = a if $A_i \in R_j$, otherwise $s_{j,i}$ = b.

Step 2: Repeat t … o change or one row is made up … "a"

Step 2.1: For each X→ Y , choo … ere the elements corresponding to X take the value a.

Step 2.2: In those chosen rows (must be at least two rows), the elements corresponding to Y also take the value a if one of the chosen rows take the value a on Y .

Testing for the lossless join property (cont)

The decomposition is *lossless* if one row is entirely made up by “a” values.

The algorithm can be found as the Algorithm 15.2 in E/N book.

Note: The correctness of the algorithm is based on the assumption that no null values are allo

If and only if exists an order such that

a superkey of R_i or M_{i-1}, where M_{i-1} is the join on R_1, R_2, … R_{i-1}

Testing for the lossless join property(cont)

Example: R = (A,B,C,D), F = {$A \rightarrow B, A \rightarrow C, C \rightarrow D$}.

Let $R_1 = (A,B,C)$, $R_2 = (C,D)$.

Initially, S is

	A	B	C	D
R_1	a	a		
R_2	b	b		

Note: rows 1 and 2 of S agree on {C}, ft hand side of $C \rightarrow D$. Therefore, change the D value on rows 1 to a, matching the value from row 2.

Now row 1 is entirely a's, so the decomposition is lossless.

(Check it.)

Testing for the lossless join property(cont)

Example 2: $R = (A,B,C,D,E)$,

$F = \{AB \rightarrow CD, A \rightarrow E, C \rightarrow D\}$. Let $R_1 = (A,B,C)$,

$R_2 = (B,C,D)$ and $R_3 = (C,D,E)$.

Example 3: $R = (A,B,C,D,E,G)$,

$F = \{A \rightarrow B, C \rightarrow DE, AB \rightarrow G\}$. Let $R_1 = (A,B)$,

$R_2 = (C,D,E)$ and $R_3 = (A,C,G)$.

Testing for the lossless join property(cont)

Example 4: R = (*A,B,C,D,E,G*),

F = {*AB* →*G, C* → *DE*, *A* → *B*}.

Let R_1 = (*A,B*), R_2 = (*C, D, E*) and R_3 = (*A,C,G*).

	A	B	C	D	E	G
R_1	a	a	b	b	b	b
R_2	b	b	a	a	a	b
R_3	a	b	a	b	b	a

	A	B	C	D	E	G
R_1	a	a	b	b	b	b
R_2	b	b	a	a	a	b
R_3	a	b	a	~~b~~ a	~~b~~ a	a

		B	C	D	E	G
R_1	a	a	b	b	b	b
R_2	b	b	a	a	a	b
R_3	a	~~b~~ a	a	a	a	a

Lossless decomposition into BCNF

Algorithm TO_BCNF

D := $\{R_1, R_2, \ldots R_n\}$

While $\exists$ a $R_i \in$ D and R_i is not in BCNF **Do**

{ find a $X \rightarrow Y$ in R_i that violates BCNF; replace R_i in D by $(R_i - Y)$ and $(X \cup Y)$; }

F = {A→B, A→C, A→D, C→E, },

R1 = (C, D, E, G), R2 = (A, B, C, D)

R11 = (C, E, G), R12 = (E, D) due E→D

R21 = (A, B, C), R22 = (C, D) because of C → D

Lossless decomposition into BCNF

Algorithm TO_BCNF

D := $\{R_1, R_2, \ldots R_n\}$

While $\exists$ a $R_i \in$ D and R_i is not in BCNF **Do**

{ find a $X \to Y$ in R_i that violates BCNF; replace R_i in D by $(R_i - Y)$ and $(X \cup Y)$; }

Since a $X \to Y$ violating BCNF is not always in F, culty is to veri R_i is in BCNF;

see the approach below:

1. For each subset X of R_i, computer X^+.

2. $X \to (X^+|_{R_i} - X)$ violates BCNF, if $X^+|_{R_i} - X \neq \emptyset$ and $R_i - X^+ \neq \emptyset$.

Here, $X^+|_{R_i} - X = \emptyset$ means that each F.D with X as the left hand side is trivial;

$R_i - X^+ = \emptyset$ means X is a superkey of R_i

Lossless decomposition into BCNF (cont)

Example: (From Desai 6.31)

Find a BCNF decomposition of the relation scheme below:

$SHIPPING(Ship, Capacity, Date, Cargo, Value)$

F consists of:

$Ship \rightarrow Capacity$

$\{Ship, Date\} \rightarrow Cargo$

$\{Cargo, Capacity\} \rightarrow Value$

Lossless decomposition into BCNF (cont)

From *Ship*$\rightarrow$ *Capacity*, we decompose *SHIPPING* into

R_1(*Ship* , *Date* , *Cargo* , *Value*)

Key: {*Ship*,*Date*}

A nontrivial FD in F^+ violates BCNF:

{*Ship* , *Cargo*}

and

R_2(*Ship* , *Capacity*)

Key: {*Ship*}

Only one nontrivial FD in F^+: *Ship* $\rightarrow$ *Capacity*

Ship $\rightarrow$ Capacity

{*Ship* , *Date*}$\rightarrow$ *Cargo*

{*Cargo* , *Capacity*}$\rightarrow$ *Value*

Lossless decomposition into BCNF (cont)

R_1 is not in BCNF so we must decompose it further into

Ship → Capacity

{*Ship* , *Date*}→ *Cargo*

{*Cargo* , *Capacity*}→ *Value*

R_{11} (*Ship* , *Date* , *Cargo*)

Key: {*Ship*,*Date*}

Only one nontrivial FD in F^+ with single attribute on the right side: {*Ship* , *Date*} →Cargo

and

R_{12} (*Ship* , *Cargo* , *Value*)

Key: {*Ship*,*Cargo*}

Only one nontrivial FD in F^+ with single attribute on the right side: {*Ship*,*Cargo*} → *Value*

This is in BCNF and the decomposition is lossless but not dependency preserving (the FD {*Capacity*,*Cargo*} → *Value*) has been lost.

Lossless decomposition into BCNF (cont)

Or we could have chosen {*Cargo* , *Capacity*} →*Value*, which would give us:

R_1 (*Ship* , *Capacity* , *Date* , *Cargo*)

Ship → Capacity
{*Ship* , *Date*}→ *Cargo*
{*Cargo* , *Capacity*}→ *Value*

Key: {*Ship*,*Date*}

A nontrivial FD in F^+ violates BCNF:

Ship → *Capacity*

and

R_2 (*Cargo* , *Capacity* , *Value*)

Key: {*Cargo*,*Capacity*}

Only one nontrivial FD in F^+ with single attribute on the right side: {*Cargo* , *Capacity*} → *Value* and then from $Ship \rightarrow Capacity$,

$R_{11}(Ship, Date, Cargo)$

Key: $\{Ship, Date\}$

Only one nontrivial FD in F^+ with single attribute

on the right side: $\{Ship, Date\} \rightarrow Cargo$

$Ship \rightarrow Capacity$
$\{Ship, Date\} \rightarrow Cargo$
$\{Cargo, Capacity\} \rightarrow Value$

And

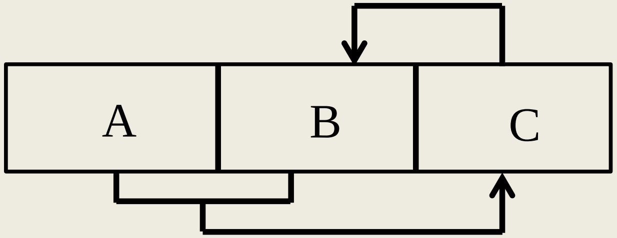
$R_{12}(Ship, Capacity$

Key: $\{Ship\}$

Only one nontrivial FD in F^+: $Ship \rightarrow C$

This is in BCNF and the decomposition is both lossless and dependency preserving.

However, there are relation schemes for which there is no lossless, dependency preserving decomposition into BCNF.



Lossless and dependency-preserving decomposition into 3NF

A lossless and dependency-preserving decomposition into 3NF is always possible.

More definitions regarding FD's are needed.

A set *F* of FD's is minimal if

1. Every FD $X \to Y$ in F is simple: *Y* consists of a single attribute,

2. Every FD $X \to A$ roper subset

   $Y \subset X$ such that

   that is, there is no $Y \subset X$ such that

$$((F - \{X \to A\}) \cup \{Y \to A\})^+ = F^+$$

Iff F |= Y → A

3. No FD in F can be removed; that is, there is no FD $X \to A$ in F such that

$$(F - \{X \to A\})^+ = F^+.$$

Iff X→A is inferred From F– { X→A}

Computing a minimum cover

F is a set of FD’s.

A *minimal cover* (or *canonical cover*) for F is a minimal set of FD’s F_{min} such that $F^+ = F^+_{min}$.

Algorithm Min_Cover

Input: a set F of functional dependencies.

Output: a minimum c

Step 1: *Reduce ri* ht to F.

Step 2: *Reduce left side*. Apply Algorith e output of Step 2.

Step 3: *Remove redundant FDs*. Apply A redundancy to the output of Step 2. The

output is a minimum cover.

Below we detail the three Steps.

Computing a minimum cover(cont)

Algorithm Reduce_right

INPUT: F.

OUTPUT: right side reduced F'.

For each FD $X \rightarrow Y \in F$ where Y = $\{A_1, A_2, \ldots, A_k\}$, we use all X $\rightarrow \{A_i\}$ (for $1 \leq i \leq k$) to replace $X \rightarrow Y$.

Algorithm Reduce_lef

INPUT: right side redu

OUTPUT: right and left side reduced F'.

For each $X \rightarrow \{A\} \in F$ where X = $\{A_i : 1 \leq i \leq k\}$, do the following. For i = 1 to k, replace X with $X - \{A_i\}$ if A $\in (X - \{A_i\})^+$.

Algorithm Reduce_redundancy

INPUT: right and left side reduced F.

OUTPUT: a minimum cover F' of F.

For each FD $X \rightarrow \{A\} \in F$, remove it from F if: $A \in X^+$ with respect to $F - \{X \rightarrow \{A\}\}$.

Example:

R = (A, B, C, D, E, G)

F = {A $\rightarrow$ BCD, B $\rightarrow$ CDE, AC $\rightarrow$ E}

Step 1: F' = {A $\rightarrow$ B, A$\rightarrow$C, A $\rightarrow$ D, B$\rightarrow$ C, B$\rightarrow$ D, B$\rightarrow$ E, AC $\rightarrow$ E}

Step 2: AC $\rightarrow$ E

C^+ = {C}; thus C $\rightarrow$ E is not inferred by F'.

Hence, AC $\rightarrow$ E cannot be replaced by A $\rightarrow$ E.

A^+ = {A, B, C, D, E}; thus, A$\rightarrow$ E is inferred by F'.

Hence, AC$\rightarrow$ E can be re

F'' = {A$\rightarrow$ B, A$\rightarrow$C, A$\rightarrow$ D, A$\rightarrow$ E, B$\rightarrow$ C, B

Step 3: $A+|_{F'' - \{A \rightarrow B\}}$ = {A, C, D, E}; thus A$\rightarrow$ by F'' −{A$\rightarrow$B}.

That is, A$\rightarrow$B is not redundant.

$A+|_{F'' - \{A \rightarrow C\}}$ = {A, B, C, D, E}; thus, A$\rightarrow$ C is redundant.

Thus, we can remove A$\rightarrow$C from F'' to obtain F'''.

Iteratively, we can A$\rightarrow$D and A$\rightarrow$E but not the others.

Thus, F_{min}={A$\rightarrow$B, B$\rightarrow$C, B$\rightarrow$D, B$\rightarrow$E}.

3NF decomposition algorithm

Algorithm 3NF decomposition

1. Find a minimum cover F' of F.

2. For each left side X that appears in F', do:

create a relati A_m where $X \rightarrow \{A_1\}, \ldots, X \rightarrow \{A_m\}$ are all the

dependencies in F' with X as left

3. if none of the relation schemas contains a key of R,

create one more relation schema that contains attributes that form a key for R.

See E/N Algorithm 15.4.

Example:

R = (A, B, C, D, E, G)

F_{min}={A→B, B→C, B→D, B→E}.

Candidate key: (A, G)

R_1 = (A, B), R_2 = (B, C, D, E)

R_3 = (A, G)

3NF decomposition algorithm (cont)

Example: (From Desai 6.31)

Beginning again with the *SHIPPING* relation. The functional dependencies already form a canonical cover.

- From *Ship*→*Capacity*, derive R_1(*Ship*, *Capacity*),

- From {*Ship*,*Date*

  R_2(<u>*Ship* , *Date*</u> ,

- From {*Capacity*,*Cargo*} → *Value*, d

  R_3(<u>*Capacity* , *Cargo*</u> , *Value*).

- There are no attributes not yet included and the original key {*Ship*,*Date*} is included in R_2.

3NF decomposition algorithm (cont)

Another Example: Apply the algorithm to the LOTS example given earlier.

A minimal cover is

$$\{ Property\_Id \rightarrow Lot\_No,$$
$$Property\_Id \rightarrow Area, \{City, Lot\_No\} \rightarrow Property\_Id,$$
$$Area \rightarrow Price, Area \rightarrow City, City \rightarrow Tax\_Rate \}.$$

This gives the decomp

$R_1(\underline{Property\_Id}, Lot\_No, Area)$

$R_2(\underline{City, Lot\_No}, Property\_Id)$

$R_3(\underline{Area}, Price, City)$

$R_4(\underline{City}, Tax\_Rate)$

Exercise 1: Check that this is a lossless, dependency preserving decomposition into 3NF.

Exercise 2: Develop an algorithm for computing a key of a table R with respect to a given *F* of FDs.

Summary

Data redundancies are undesirable as they create the potential for update anomalies,

One way to remove such redundancies is to normalise a design, guided by FD's.

BCNF removes all redundancies due to FD's, but a dependency preserving decomp

A dependency preserving, lossless de to 3NF can always be found, but some redundancies may r

Even where a dependency preserving, lossless decomposition that removes all redundancies can be found, it may not be possible, for efficiency reasons, to remove all redundancies.