Test Lossless Join property

This previous test works on **binary** decompositions, below is the general solution to testing lossless join property

Algorithm TEST_LJ:

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.
2. Repeat the following process until (1) S has no change OR (2) one row is made up entirely of "a" symbols.
   i. For each X→ Y , choose the rows where the elements corresponding to X take the value a.
   ii. 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 .

Verdict: Decomposition is *lossless* if one row is entirely made up by "a" values.

Testing lossless join property(cont)

Example 1:
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).

	A	B	C	D
R_1	a	a	a	b
R_2	b	b	a	a

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

CHEAT SHEET: Algorithm TEST_LJ

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.
2. Repeat the following process till S has no change or one row is made up entirely of “a” symbols.
   1. For each $X \rightarrow Y$, choose the rows where the elements corresponding to X take the value a.
   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 lossless join property(cont)

Example 1:
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).

	A	B	C	D
R_1	a	a	a	~~b~~ a
R_2	b	b	a	a

Note: rows 1 and 2 of S agree on {C}, which is the left-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.

CHEAT SHEET: Algorithm TEST_LJ

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.
2. Repeat the following process till S has no change or one row is made up entirely of "a" symbols.
   1. For each X→ Y , choose the rows where the elements corresponding to X take the value a.
   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 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
$R_3 = (C,D,E)$.

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

CHEAT SHEET: Algorithm TEST_LJ

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.
2. Repeat the following process till S has no change or one row is made up entirely of "a" symbols.
   1. For each $X \rightarrow Y$, choose the rows where the elements corresponding to X take the value a.
   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 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
$R_3 = (C,D,E)$.

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

Not lossless join

CHEAT SHEET: Algorithm TEST_LJ

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 $A_j \in R_j$, otherwise $s_{j,i}$ = b.
2. Repeat the following process till S has no change or one row is made up entirely of “a” symbols.
    1. For each X→ Y , choose the rows where the elements corresponding to X take the value a.
    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 lossless join property(cont)

Example 3:

R = (A,B,C,D,E,G),
F = {$C \to DE$, $A \to B$, $AB \to G$}.
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	⟵

CHEAT SHEET: Algorithm TEST_LJ

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.
2. Repeat the following process till S has no change or one row is made up entirely of "a" symbols.
   1. For each $X \to Y$, choose the rows where the elements corresponding to X take the value a.
   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 lossless join property(cont)

Example 3:

R = (A,B,C,D,E,G),
F = {$C \to DE$, $A \to B$, $AB \to G$}.
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~~ a	~~b~~ a	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

CHEAT SHEET: Algorithm TEST_LJ

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.
2. Repeat the following process till S has no change or one row is made up entirely of "a" symbols.
   1. For each $X \to Y$, choose the rows where the elements corresponding to X take the value a.
   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 lossless join property(cont)

Example 3:

R = (A,B,C,D,E,G),
F = {$C \rightarrow DE$, $A \rightarrow B$, $AB \rightarrow G$}.
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	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 join

CHEAT SHEET: Algorithm TEST_LJ

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.
2. Repeat the following process till S has no change or one row is made up entirely of “a” symbols.
   1. For each $X \rightarrow Y$, choose the rows where the elements corresponding to X take the value a.
   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 .

Checkpoint

Previous:

1. The test for lossless join property
2. The dependency preservation property

Next:

1. The method to decompose to BCNF and 3NF
2. Minimal Cover and Equivalence
3. The method to decompose to 3NF

Testing for BCNF

Testing of a relation schema *R* to see if it satisfies BCNF can be simplified in **some** cases (but **not** all cases):

- To check if a nontrivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF, compute α^+ (the attribute closure of α), and verify that it includes all attributes of *R*; that is, it is a superkey for *R*.
- To check if a relation schema *R* is in BCNF, it suffices to check only the dependencies in the given set *F* for violation of BCNF, rather than check all dependencies in F^+.

Testing for BCNF

NOTE: We cannot use F to test relations R_i (decomposed from R) for violation of BCNF. It may not suffice.

Consider R(A, B, C, D, E) with F = {A -> B, BC -> D}.

Suppose R is decomposed into R1 = (A, B) and R2 = (A, C, D, E).

Neither of the dependencies in F contains only attributes from R2.
So R2 is in BCNF? No, AC -> D is in F+.

Example above : X →Y violating BCNF is not always in F.
It passing with respect to the projection of F on R_i

Testing Decomposition for BCNF

An alternative BCNF test is sometimes easier than computing every dependency in $F+$.

To check if a relation schema R_i in a decomposition of R is truly in BCNF, we apply this test:

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

- $X \rightarrow (X^+|_{Ri} - X)$ violates BCNF, if $X^+|_{Ri} - X \neq \emptyset$ and $R_i - X^+ \neq \emptyset$.
- This will show if R_i violates BCNF.

Explanation:

- $X^+|_{Ri} - X = \emptyset$ means 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

Algorithm TO_BCNF

- D := {$R_1, R_2, \ldots R_n$}
- **While** (there exists a $R_i \in$ D and R_i is not in BCNF) **Do**

1 . find a X $\rightarrow$Y in R_i that violates BCNF;

2. replace R_i in D by ($R_i - Y$) and (X $\cup$ Y);

Lossless Decomposition into BCNF

Example:

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*

We know this relation is not in BCNF

Algorithm TO_BCNF

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

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

1 . find a X $\rightarrow$Y in R_i that violates BCNF;

2. replace R_i in *D* by (R_i − Y) and (X ∪ Y);

Lossless Decomposition into BCNF (V1)

From *Ship* → *Capacity*, we decompose *SHIPPING* into R_{1A} and R_{2A}

R_{1A}(*Ship* , *Date* , *Cargo* , *Value*) with Key: {*Ship*,*Date*}

A nontrivial FD in F^+ violates BCNF: {*Ship* , *Cargo*} → Value

and

R_{2A}(*Ship* , *Capacity*) with Key: {*Ship*}

Only one nontrivial FD in F^+: *Ship* → *Capacity*

SHIPPING (*Ship* , *Capacity* , *Date* , *Cargo* , *Value*)
F consists of: Ship → Capacity, {*Ship* , *Date*}→ *Cargo*, {*Cargo* , *Capacity*}→ *Value*

Lossless Decomposition into BCNF (V1)

R_1 is not in BCNF so we must decompose it further into R_{11A} and R_{12A}

R_{11A} (*Ship* , *Date* , *Cargo*) with Key: {*Ship*,*Date*}

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

and

R_{12A} (*Ship* , *Cargo* , *Value*) with 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.

SHIPPING (*Ship* , *Capacity* , *Date* , *Cargo* , *Value*)
F consists of: Ship → Capacity, {*Ship* , *Date*}→ *Cargo*, {*Cargo* , *Capacity*}→ *Value*

Lossless Decomposition into BCNF (V2)

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

R_{1B} (*Ship* , *Capacity* , *Date* , *Cargo*) with Key: {*Ship*,*Date*}

A nontrivial FD in F^+ violates BCNF: *Ship* → *Capacity*

and

R_{2B} (*Cargo* , *Capacity* , *Value*) with Key: {*Cargo*,*Capacity*}

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

Once again, R_{1B} is not in BCNF so we must decompose it further…

SHIPPING (*Ship* , *Capacity* , *Date* , *Cargo* , *Value*)
F consists of: Ship → Capacity, {*Ship* , *Date*}→ *Cargo*, {*Cargo* , *Capacity*}→ *Value*

Lossless Decomposition into BCNF (V2)

R_1 is not in BCNF so we must decompose it further into R_{11B} and R_{12B}

R_{11B} (*Ship* , *Date* , *Cargo*) with Key: {*Ship*,*Date*}

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

and

R_{12B} (*Ship* , *Capacity*) with Key: {*Ship*}

Only one nontrivial FD in F^+: *Ship* → *Capacity*

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

SHIPPING (*Ship* , *Capacity* , *Date* , *Cargo* , *Value*)
F consists of: Ship → Capacity {*Ship* , *Date*}→ *Cargo*, {*Cargo* , *Capacity*}→ *Value*

Lossless Decomposition into BCNF

With this algorithm from the previous slide…

We get a decomposition *D* of *R* that does the following:

- May **not** preserves dependencies
- Has the lossless join property
- Is such that each resulting relation schema in the decomposition is in BCNF

Lossless decomposition into BCNF

Review: 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 $(X \cup Y)$; }

Since a $X \rightarrow Y$ violating BCNF is not always in F, the main difficulty is to verify if R_i is in BCNF;

Computing a Minimal Cover (Step 1)

Step 1: Reduce Right: 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 $X \rightarrow Y$.

Practice:

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

F = {A ->BCD, B -> CDE, AC -> E}

At the end of step 1 we have : F’ = {A -> B, A -> C, A -> D, B -> C, B -> D, B -> E, AC -> E}

Computing a Minimal Cover (Step 2)

Step 2: Reduce Left: 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 $A \in (X - \{A_i\})^+$.

From Step 1, we had: F' = {A -> B, A -> C, A -> D, B -> C, B -> D, B -> E, AC -> E}

AC -> E

C^+ = {C}; thus C -> E is not inferred by F'.

Hence, AC -> E cannot be replaced by C -> E.

A^+ = {A, B, C, D, E}; thus, A -> E is inferred by F'.

Hence, AC -> E can be replaced by A -> E.

We now have F'' = {A -> B, A -> C, A -> D, A -> E, B -> C, B -> D, B -> E}

Computing a Minimal Cover (Step 3)

Step 3: Reduce_redundancy: For each FD $X \rightarrow \{A\} \in F$, remove it from F if: $A \in X^+$ with respect to $F - \{X \rightarrow \{A\}\}$.

From Step 2, we had: F'' = {A -> B, A -> C, A -> D, A -> E, B -> C, B -> D, B -> E}

$A+|_{F'' – \{A \text{ -> } B\}}$ = {A, C, D, E}; thus A -> B is not inferred by F'' – {A -> B}.
That is, A -> B is not redundant.
$A+|_{F'' – \{A \text{ -> } C\}}$ = {A, B, C, D, E}; thus, A -> C is redundant.
Thus, we can remove A -> C from F'' to obtain F'''.

We find that we can remove A -> D and A -> E but not the others.
Thus, F_{min}={A -> B, B -> C, B -> D, B -> E}.

3NF Decomposition Algorithm

Algorithm 3NF decomposition

1. Find a minimal cover *G* for *F*.
2. For each left-hand-side *X* of a functional dependency that appears in *G*, create a relation schema in *D* with attributes $\{X \cup \{A_1\} \cup \{A_2\} \ldots \cup \{A_k\}\}$, where $X \rightarrow A_1$, $X \rightarrow A_2$, ..., $X \rightarrow A_k$ are the only dependencies in *G* with *X* as left-hand-side (*X* is the key to this relation).
3. If none of the relation schemas in *D* contains a key of *R*, then create one more relation schema in *D* that contains attributes that form a key of *R*.
4. Eliminate redundant relations from the resulting set of relations in the relational database schema. A relation *R* is considered redundant if *R* is a projection of another relation *S* in the schema; alternately, *R* is subsumed by *S*.

3NF Decomposition Algorithm

With this algorithm from the previous slide…

We get a decomposition *D* of *R* that does the following:

- Preserves dependencies
- Has the nonadditive (lossless) join property
- Is such that each resulting relation schema in the decomposition is in 3NF

3NF Decomposition Algorithm

Example ONE:

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

$F_{min}=\{A\text{->}B, B\text{->}C, B\text{->}D, B\text{->}E\}$.

Candidate key: (A, G)

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

$R_3 = (A, G)$

3NF Decomposition Algorithm

Example TWO:

Following from the *SHIPPING* relation. The functional dependencies already form a canonical cover.

- From *Ship*→*Capacity*, derive R_1(*Ship*, *Capacity*),
- From {*Ship*,*Date*} → *Cargo*, derive R_2(*Ship* , *Date* , *Cargo*),
- From {*Capacity*,*Cargo*} → *Value*, derive R_3(*Capacity* , *Cargo* , *Value*).
- There are no attributes not yet included and the original key {*Ship*,*Date*} is included in R_2.

SHIPPING (*Ship* , *Capacity* , *Date* , *Cargo* , *Value*)
F consists of: Ship → Capacity, {*Ship* , *Date*}→ *Cargo*, {*Cargo* , *Capacity*}→ *Value*

3NF Decomposition Algorithm

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

One possible minimal cover is

{ Property_Id→Lot_No,

Property_Id → Area, {City,Lot_No} → Property_Id,

Area → Price, Area → City, City → Tax_Rate }.

This gives the decomposition:

R_1 (<u>Property_Id</u> , Lot_No , Area)

R_2 (<u>City , Lot_No</u> , Property_Id)

R_3 (<u>Area</u> , Price , City)

R_4 (<u>City</u> , Tax_Rate)