CS354: Database

Recap: Last Class

- Normal form: set of properties that relations must satisfy
  - Relations exhibit less anomalies
  - Successively higher degrees of stringency
- 1NF: most basic normal form with atomic attributes
- Functional dependencies: X —> Y
  - Armstrong's axioms to derive additional FDs to find good relational decompositions

Finding Keys of Relation R

- Bad news: find all keys of a relation is NP-complete
  - Running time of algorithm to solve the problem exactly is exponentially increasing with the problem size
  - Large NP-complete problems are difficult to solve!
  - No efficient solution to find all the keys
  - Brute force algorithm: Check every subset of attributes for super key strategy — tests every possible solution
- Solution: use heuristics to find all the keys of a relation
  - Turn towards closures to help us find keys in a relation

Attribute Closure Set

- If X is an attribute set, the closure X^+ is the set of all attributes B such that X —> B
    - X is subset of X^+ since X —> X
    - X^+ includes all attributes that are functionally determined from X
- Importance: If $X^+ = R$, then X is a superkey
    - Closure can tell us if set of attributes X is a superkey

Example: Closure

- **Product**(name, category, color, department, price)
  - name —> color
  - category —> department
  - color, category —> price
- Attribute Closure:
  - $\{name\}^+ = \{name, color\}$
  - $\{name, category\}^+ = \{name, color, category, department, price\}$

Finding a Key after Closure

- If X^+ not equal to the relation, we must augment more attributes to X to obtain a key
- If $X^+ = R$, then X is superkey — check for minimality
  - Remove one or more attributes A
  - Compute the closure of X - A to see if $(X - A)^+ = R$
  - X is a key if $(X - A)^+$ not equal R for any attribute A

Closure Algorithm

- Input: A set F of FDs on a relation schema R, and a set of attributes X, which is a subset of R

- Algorithm:
  Initialize $X^+ := X$
  repeat
      old $X^+ := X^+$
      for each functional dependency $Y \longrightarrow Z$ in F
          if X^+ superset Y, then $X^+ := X^+$ union Z
  until (X^+ = old X^+)

Example: Closure Algorithm

EmpProj(SSN, FName, LName, PNo, PName, PLocation, Hours)

- SSN —> FName, LName
- PNo —> PName, PLocation
- SSN, PNo —> Hours

Example: Closure Algorithm (2)

- Initialize SSN^+ := SSN
- Repeat loop (for each FD)
  - SSN —> FName, LName
    => SSN^+ := SSN, FName, LName
  - PNo —> PName, PLocation
    => no change
  - SSN, PNo —> Hours
    => no change
- Result: SSN^+ := SSN, FName, LName

Since there were changes, repeat another loop through FDs, which results in no changes => done

Example: Closure Algorithm (3)

- Initialize PNo^+ := PNo
- Repeat loop (for each FD)
  - SSN —> FName, LName
    => no change
  - PNo —> PName, PLocation
    => PNo^+ := PNo, PName, PLocation
  - SSN, PNo —> Hours
    => no change
- Result: PNo^+ := PNo, PName, PLocation

Since there were changes,
repeat another loop through **FDs**, which results in no changes
=> done

Example: Closure Algorithm (4)

- Initialize $(SSN, PNo)^+$:= SSN, PNo
- Repeat loop (for each FD)
  - SSN —> FName, LName
    => $(SSN, PNo)^+$:= SSN, PNo, FName, LName
  - PNo —> PName, PLocation
    => $(SSN, PNo)^+$:= SSN, PNo, FName, LName, PName, PLocation
  - SSN, PNo —> Hours
    => $(SSN, PNo)^+$:= SSN, PNo, FName, LName, PName, PLocation, Hours
- Result: $(SSN, PNo)^+$:= SSN, PNo, FName, LName, PName, PLocation, Hours

Example: Closure Algorithm (4)

- Summary of results:
  - SSN^+ := SSN, FName, LName
  - PNo^+ := PNo, PName, PLocation
  - $(SSN, PNo)^+$:= SSN, PNo, FName, LName, PName, PLocation, Hours
- (SSN, PNo) is a superkey!
- (SSN, PNo) is minimal superkey
  - $\{(SSN, PNo) - (SSN)\}^+ = (PNo)^+$
  - $\{(SSN, PNo) - (PNo)\}^+ = (SSN)^+$

Finding Keys: Heuristic 1

- Increase/decrease until you find keys
- Step 1: Compute closure of all functional dependencies in F
- Step 2:
    - If deficient, then add missing attributes to the LHS until the closure is equal to the relation
    - If sufficient, then remove extraneous attributes from the LHS until set is minimal

Example: Key Heuristic 1

- **R**(A, B, C, D, E, F)
  - A —> B, C
  - B, D —> E, F
  - F —> A
- Step 1: Closure of all functional dependencies
  - A^+ = A, B, C
  - $(B, D)^+$ = A, B, C, D, E, F
  - F^+ = F, A, B, C

Example: Key Heuristic 1 (2)

- Step 2: Insert / remove attributes
  - A^+ = A, B, C — insufficient so add
    - Add D: $(A, D)^+$ = A, B, C, D, E, F —> key!
    - Add E: $(A, E)^+$ = A, B, C, E
    - Add F: $(A, F)^+$ = A, B, C, F
    - Add E, F: $(A, E, F)^+$ = A, B, C, E, F
    - No more so done

Example: Key Heuristic 1 (3)

- Step 2: Insert / remove attributes
  - $(B, D)^+ = A, B, C, D, E, F$ — sufficient so try deleting
    - Delete B: $(D)^+ = D$
    - Delete D: $(B)^+ = B$
    - No more so done

B, D is minimal and thus a key!

Example: Key Heuristic 1 (4)

- Step 2: Insert / remove attributes
  - F^+ = F, A, B, C — insufficient so add
    - Add D: $(D, F)^+$ = A, B, C, D, E, F —> key!
    - Add E: $(E, F)^+$ = A, B, C, E, F
    - No more so done

Keys are: (A, D), (B, D), and (D, F)!

Finding Keys: Heuristic 2

- Find necessary attributes first
- Find the irreplaceable attributes
    - Attribute is replaceable if it appears in the RHS of some functional dependency
- A key must include every irreplaceable attribute
- Base set is set of all irreplaceable attributes
- Add other attributes to base set until you have a key

Example: Key Heuristic 2

- **R**(A, B, C, D, E, F)
  - A —> B, C
  - B, D —> E, F
  - F —> A
- Step 1: Find irreplaceable attributes and construct base set
  Base set = {D}

Example: Key Heuristic 2 (2)

- Step 2: Add other attributes until you have key
  - Add A: $(A, D)^+ = A, B, C, D, E, F$ —> key!
  - Add B: $(B, D)^+ = A, B, C, D, E, F$ —> key!
  - Add C: $(C, D)^+ = C, D$
  - Add E: $(D, E)^+ = D, E$
  - Add F: $(D, F)^+ = A, B, C, D, E, F$ —> key!

Example: Key Heuristic 2 (3)

- Step 2: Add other attributes until you have key (do not expand known keys)
  - Add C: $(C, D, E)^+ = C, D, E$
  - No more to add, so done!

Second Normal Form (2NF)

- (Definition) A relation schema R is in 2NF if every non-prime attribute (i.e., not a member of any candidate key) A in R is not partially dependent on any key of R
    - Relation is 1NF (attributes are atomic)
    - No non-key attribute that is functionally determined by only a (proper) subset of a key

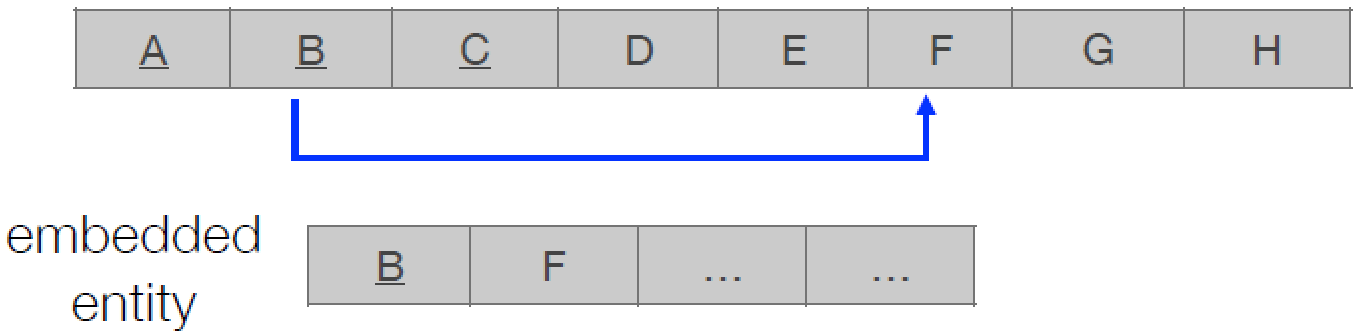
A	B	C	D	E	F	G	H

key
(A, B, C)

B —> F means F is functionally dependent on subset of key => violation of 2NF

2NF Meaning

A relation that violates 2NF contains another embedded autonomous entity



Example: Violation of 2NF

- **EmpProj**(<u>SSN</u>, FName, LName, <u>PNo</u>, PName, Hours)
    - SSN —> FName, LName
    - PNo —> PName
    - SSN, PNo —> Hours
- FName is not part of any key
- SSN is (proper) subset of a key
- Violation since Employee entity is embedded (SSN, FName, LName)

Decomposition for Normal Form Violations

- Break a relation into two or more relations
- One possibility for **EmpProj**(<u>SSN</u>, FName, LName, <u>PNo</u>, PName, Hours):
  - R1(PNo, PName, Hours)
  - R2(SSN, FName, Lname)
- Another possibility for EmpProj
  - R3(SSN, FName, Lname)
  - R4(SSN, PNo, PName, Hours)

Are these good or bad decompositions?

Decomposition Effect

- Populate the new relations using data of the original relation
  - Achieve this by using projection operation on the original relation
  - Example:

$$R1 = \pi_{\mathrm{SSN,FName,LName}}(\mathrm{EmpProj})$$
$$R2 = \pi_{\mathrm{PNo,PName,Hours}}(\mathrm{EmpProj})$$

Decomposition Effect (2)

- Can we obtain the same information stored in the original relation?

- Reconstruction algorithm:
  If ($R1 \cap R2 \neq \emptyset$) {
  reconstruction = R1 * R2 // Natural join
  } else {
  reconstruction = R1 x R2 // Cartesian product
  }

Example: Decomposition Effect

SSN	FName	LName	PNo	PName	Hours
111-11-1111	John	Smith	pj1	ProjectX	20
111-11-1111	John	Smith	pj2	ProjectY	10
333-33-3333	Jack	Rabbit	pj1	ProjectX	5

SSN	FName	LName
111-11-1111	John	Smith
333-33-3333	Jack	Rabbit

PNo	PName	Hours
pj1	ProjectX	20
pj2	ProjectY	10
pj1	ProjectX	5

Example: Reconstructing After Decomposition

SSN	FName	LName
111-11-1111	John	Smith
333-33-3333	Jack	Rabbit

X

PNo	PName	Hours
pj1	ProjectX	20
pj2	ProjectY	10
pj1	ProjectX	5

SSN	FName	LName	PNo	PName	Hours
111-11-1111	John	Smith	pj1	ProjectX	20
111-11-1111	John	Smith	pj2	ProjectY	10
111-11-1111	John	Smith	pj1	ProjectX	5
333-33-3333	Jack	Rabbit	pj1	ProjectX	20
333-33-3333	Jack	Rabbit	pj2	ProjectY	10
333-33-3333	Jack	Rabbit	pj1	ProjectX	5

Extraneous tuples that weren't present in original relation!

Decomposition Relation Requirements

- Must be able to obtain all tuples in the original relation R using the reconstruction algorithm
  - Missing tuples means that we have lost information which is unacceptable
- Must not obtain extraneous tuples that were not present in the original relation R using the reconstruction algorithm
  - Invalid information in the relation which is also unacceptable

Lossless Decomposition

- A decomposition of relation R into 2 relations R1 and R2 is called lossless if and only if
content(R1) * content(R2) = content(R) or
content (R1) x content(R2) = content(R)

- 2 lemmas that provide needed guidelines to decompose R to guarantee lossless

    - Lemma 1: $\text{content}(R) \subseteq \text{content}(R_1) * \text{content}(R_2)$

    - Lemma 2: If either $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$, then $\text{content}(R) = \text{content}(R_1) * \text{content}(R_2)$

Example: 2NF via Lemma 2

- **EmpProj**(<u>SSN</u>, FName, LName, <u>PNo</u>, PName, Hours)
  - SSN —> FName, LName
  - PNo —> PName
  - SSN, PNo —> Hours
- At least one violating FD
  - SSN —> FName
  - SSN —> LName

Remove all attributes functionally dependent on SSN => compute closure of SSN

Example: 2NF via Lemma 2 (2)

- R1(SSN$^+$) = **R1**(SSN, FName, LName)
- R2(R - R1) = **R2**(PNo, PName, Hours)
  - To satisfy lemma 2, add SSN to R2 => **R2**(SSN, PNo, PName, Hours)
  - R1 $\cap$ R2 = SSN, and SSN —> R1

Are R1 and R2 in the 2NF?

Example: 2NF via Lemma 2 (3)

- **R1**(SSN, FName, LName)
  - SSN —> FName, FName — key = good dependency
- **R2**(SSN, PNo, PName, Hours)
  - SSN, PNo —> Hours — key = good dependency
  - PNo —> PName — not key = bad!

Remove all attributes functionally dependent on PNo => compute closure of PNo

Example: 2NF via Lemma 2 (4)

- R21(PNo^+) = **R21**(<u>PNo</u>, PName)
- R22(R2 - R21) = **R22**(<u>SSN</u>, Hours)
    - To satisfy lemma 2, add PNo to R22 =>
      R22(SSN, PNo, Hours)
- Resulting decomposition:
  R1(<u>SSN</u>, FName, LName)
  R21(<u>PNo</u>, PName)
  R22(<u>SSN</u>, <u>PNo</u>, Hours)

Are R1, R21, and R22 in the 2NF?

Example: 2NF Complaint

- **Employee2**(<u>SSN</u>, FName, LName, DNo, DName, MgrSSN)
  - SSN —> FName, LName, DNo
  - DNo —> DName, MgrSSN
- Employee2 is 2NF as DNo is not a subset of any key and neither of the functional dependencies violate 2NF criteria
- But…
  - Insert anomaly — adding new department results in NULL values
  - Delete anomaly — deleting an employee may delete information about department
  - Update anomaly — changing department name results in updates of multiple tuples

Transitive Functional Dependency

A functional dependency A —> B is a transitive functional dependency in relation R if there is a set of attributes X such that:

- A —> X
- X —> B
- X is not a super key

Third Normal Form (3NF)

(Definition) A relation schema R is in 3NF if, whenever a nontrivial functional dependency X —> A holds in R, either (a) X is a super key of R, or (b) A is a prime attribute of R

- R is in 2NF
- Every non-key attribute is non-transitively dependent on all the keys

<u>A</u>	<u>B</u>	<u>C</u>	D	E	F	G	H

key (A, B, C)

If E —> G, then transitive dependency

(A, B, C) —> E —> G

Example: 3NF Violation

- **Employee2**(<u>SSN</u>, FName, LName, DNo, DName, MgrSSN)
    - SSN —> FName, LName, DNo
    - DNo —> DName, MgrSSN
- Since DNo is not a super key, there is a transitive dependency SSN —> DNo —> DName, MgrSSN

Simpler Form of 3NF

- A relation R is 3NF if and only if for every functional dependency X —> B in relation R, one of the following must be true:
  - X is a superkey, or
  - B is a key attribute (part of some key)
- Violation detection: Check every functional dependency X —> B for:
  - B is a non-key attribute, and
  - X is not a superkey

Example: 3NF Violation Take 2

Employee2(SSN, FName, LName, DNo, DName, MgrSSN)

- SSN —> FName, LName, DNo
  - FName, LName, and DNO are non-key attributes => YES
  - SSN is not superkey => NO
  - FD is good
- DNo —> DName, MgrSSN
  - Name and MgrSSN are non-key attributes => YES
  - DNo is not superkey => YES
  - FD is bad and a 3NF violation

Example: 3NF Decomposition

- Solution: remove the violation by removing X^+ from the original relation
- R(A, B, C, D, E, F)
    - A —> B, C, D
    - D —> E, F
- Step 1: Find all keys
    - $A^+ = (A, B, C, D, E, F)$

Example: 3NF Decomposition (2)

- Step 2: Is R 2NF?
  - Key(s): A
  - Non-key attributes: B, C, D, E, F
  - Is any of the non-key attributes functionally dependent on subset of (A)? NO
  - Relation is 2NF

Example: 3NF Decomposition (3)

- Step 3: Is R 3NF?
  - Key(s): A
  - Non-key attributes: B, C, D, E, F
  - Is any of the non-key attributes functionally dependent on attributes that are not super key? YES!
    - D —> E, F where D is not a superkey

Example: 3NF Decomposition (4)

- Step 4: Extract offending functional dependence
  - $D^+ = (D, E, F)$
  - R1(<u>D</u>, E, F)
    R2(<u>A</u>, B, C, D)
- Step 5: Check the new relations if they are 3NF?
  - R1: D —> E, F doesn't violate 3NF criteria
  - R2: A —> B, C, D doesn't violate 3NF criteria

Summary of 1NF, 2NF, 3NF

Normal Form	Test	Normalization (Remedy)
1NF	Relation should have no multi-valued attributes or nested relations	Form new relation for each multivalued attribute or nested relation
2NF	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key	Decompose and set up a new relation for each partial key with its dependent attributes using lossless decomposition
3NF	Relation should not have a nonkey attribute functionally determined by another nonkey attribute	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attributes

Boyce-Codd Normal Form (BCNF)

(Definition) A relation schema R is in BCNF if whenever a nontrivial functional dependency X —> A holds in R, then X is a superkey of R

- Difference from 3NF: 3NF allows A to be prime attribute
- Every relation in BCNF is also in 3NF
- Most relation schemas that are in 3NF are also BCNF but not all
  - Example: R($\underline{A}$, $\underline{B}$, C)
    - A, B —> C
    - C —> A

Example: BCNF Violation

- TSS(Teacher, Subject, Student)
  - Student, Subject —> Teacher
  - Teacher —> Subject
- Keys in TSS
  - (Student, Subject)
  - (Student, Teacher)

TEACH

STUDENT	COURSE	INSTRUCTOR
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe

Example: BCNF Violation (2)

- Is TSS in the 3NF?
  - Student, Subject —> Teacher — superkey = okay
  - Teacher —> Subject
    - Is teacher a superkey? NO
    - Is subject a key attribute (part of key)? YES — okay
- Even though TSS is 3NF…
  - Duplicate information is stored in relation (teacher, subject)

Example: BCNF Violation (3)

- Problem arises when 2 or more composite keys are in a relation
- Is relation BCNF?
  - Student, Subject —> Teacher — superkey = okay
  - Teacher —> Subject
    Teacher is not a superkey => BCNF violation!
- Solution: Decompose the violating FD
  - T1(Teacher, Subject)
    R2(Teacher, Student)

Decomposition of a relation schema

If R doesn't satisfy a particular normal form,
we decompose R into smaller schemas

What's a decomposition?

$R = (A_1, A_2, \ldots, A_n)$

$D = (R_1, R_2, \ldots, R_k)$ st $R_i \subseteq R$ and $R = R_1 \cup R_2 \cup \ldots \cup R_k$

(R_i's need not be disjoint)

Replacing R by $R_1, R_2, \ldots, R_k$ – process of decomposing R

Ex: gradeInfo (rollNo, studName, course, grade)

R_1: gradeInfo (<u>rollNo, course</u>, grade)

R_2: studInfo (<u>rollNo</u>, studName)

Desirable Properties of Decompositions

Not all decomposition of a schema are useful

We require two properties to be satisfied

(i) Lossless join property

- the information in an instance r of R must be preserved in the instances $r_1, r_2, \ldots, r_k$ where $r_i = \pi_{R_i}(r)$

(ii) Dependency preserving property

- if a set F of dependencies hold on R it should be possible to enforce F by enforcing appropriate dependencies on each r_i

Lossless join property

F – set of FDs that hold on R

R – decomposed into $R_1, R_2, \ldots, R_k$

Decomposition is *lossless* wrt F if

for every relation instance r on R satisfying F,

$$r = \pi_{R_1}(r) * \pi_{R_2}(r) * \ldots * \pi_{R_k}(r)$$

Lossless joins are also called non-additive joins

$R = (A, B, C)$; $R_1 = (A, B)$; $R_2 = (B, C)$

r:

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_1	c_3

r_1:

A	B
a_1	b_1
a_2	b_2
a_3	b_1

r_2:

B	C
b_1	c_1
b_2	c_2
b_1	c_3

$r_1 * r_2$:

Original info is distorted

A	B	C
a_1	b_1	c_1
a_1	b_1	c_3
a_2	b_2	c_2
a_3	b_1	c_1
a_3	b_1	c_3

Lossy join

Spurious tuples

Testing for lossless decomposition property(1/6)

R – given schema with attributes $A_1, A_2, \ldots, A_n$
F – given set of FDs
D – $\{R_1, R_2, \ldots, R_m\}$ given decomposition of R

Is D a lossless decomposition?

Create an $m \times n$ matrix S with columns labeled as $A_1, A_2, \ldots, A_n$
and rows labeled as $R_1, R_2, \ldots, R_m$

Initialize the matrix as follows:
set $S(i,j)$ as symbol b_{ij} for all i,j.
if A_j is in the scheme R_i, then set $S(i,j)$ as symbol a_j, for all i,j

Testing for lossless decomposition property(2/6)

After S is initialized, we carry out the following process on it:

repeat
    for each functional dependency $U \rightarrow V$ in F **do**
        for all rows in S which agree on U-attributes **do**
            make the symbols in each V- attribute column
              the *same* in all the rows as follows:
                if any of the rows has an "*a*" symbol for the column
                    set the other rows to the same "*a*" symbol in the column
                else // if no "*a*" symbol exists in any of the rows
                    choose one of the "*b*" symbols that appears
                    in one of the rows for the V-attribute and
                    set the other rows to that "b" symbol in the column
until no changes to S

At the end, if there exists a row with all "*a*" symbols then D is lossless otherwise D is a lossy decomposition

Testing for lossless decomposition property(3/6)

R = (rollNo, name, advisor, advisorDept, course, grade)

FD's = { rollNo → name; rollNo → advisor; advisor → advisorDept
rollNo, course → grade}

D : { R_1 = (rollNo, name, advisor), R_2 = (advisor, advisorDept),
R_3 = (rollNo, course, grade) }

Matrix S : (Initial values)

	rollNo	name	advisor	advisor Dept	course	grade
R_1	a_1	a_2	a_3	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	b_{25}	b_{26}
R_3	a_1	b_{32}	b_{33}	b_{34}	a_5	a_6

Testing for lossless decomposition property(4/6)

R = (rollNo, name, advisor, advisorDept, course, grade)

FD's = { rollNo → name; rollNo → advisor; advisor → advisorDept
rollNo, course → grade}

D : { R_1 = (rollNo, name, advisor), R_2 = (advisor, advisorDept),
R_3 = (rollNo, course, grade) }

Matrix S : (After enforcing rollNo → name & rollNo → advisor)

	rollNo	name	advisor	advisor Dept	course	grade
R_1	a_1	a_2	a_3	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	b_{25}	b_{26}
R_3	a_1	$\cancel{b_{32}}a_2$	$\cancel{b_{33}}a_3$	b_{34}	a_5	a_6

Testing for lossless decomposition property(5/6)

R = (rollNo, name, advisor, advisorDept, course, grade)

FD's = { rollNo → name; rollNo → advisor; advisor → advisorDept
rollNo, course → grade}

D : { R_1 = (rollNo, name, advisor), R_2 = (advisor, advisorDept),
R_3 = (rollNo, course, grade) }

Matrix S : (After enforcing advisor → advisorDept)

	rollNo	name	advisor	advisor Dept	course	grade
R_1	a_1	a_2	a_3	$\cancel{b_{14}}a_4$	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	b_{25}	b_{26}
R_3	a_1	$\cancel{b_{32}}a_2$	$\cancel{b_{33}}a_3$	$\cancel{b_{34}}a_4$	a_5	a_6

No more changes. Third row with all *a* symbols. So a lossless join.

Testing for lossless decomposition property(6/6)

R – given schema. F – given set of FDs

The decomposition of R into R_1, R_2 is lossless wrt F if and only if either $R_1 \cap R_2 \rightarrow (R_1 - R_2)$ belongs to F^+ or

$R_1 \cap R_2 \rightarrow (R_2 - R_1)$ belongs to F^+

Eg. gradeInfo (rollNo, studName, course, grade)

with FDs = {rollNo, course $\rightarrow$ grade; studName, course $\rightarrow$ grade; rollNo $\rightarrow$ studName; studName $\rightarrow$ rollNo}

decomposed into

grades (rollNo, course, grade) and studInfo (rollNo, studName)

is lossless because

rollNo $\rightarrow$ studName

A property of lossless joins

D_1: $(R_1, R_2, \ldots, R_K)$ lossless decomposition of R wrt F

D_2: $(R_{i1}, R_{i2}, \ldots, R_{ip})$ lossless decomposition of R_i wrt $F_i = \pi_{R_i}(F)$

Then

$D = (R_1, R_2, \ldots, R_{i-1}, R_{i1}, R_{i2}, \ldots, R_{ip}, R_{i+1}, \ldots, R_k)$ is a lossless decomposition of R wrt F

This property is useful in the algorithm for BCNF decomposition

Dependency Preserving Decompositions

Decomposition $D = (R_1, R_2, \ldots, R_k)$ of schema R *preserves* a set of dependencies F if

$$(\pi_{R_1}(F) \cup \pi_{R_2}(F) \cup \ldots \cup \pi_{R_k}(F))^+ = F^+$$

Here, $\pi_{R_i}(F) = \{ (X \rightarrow Y) \in F^+ \mid X \subseteq R_i, Y \subseteq R_i \}$ (called projection of F onto R_i)

Informally, any FD that logically follows from F must also logically follow from the union of projections of F onto R_i's Then, D is called dependency preserving.

Testing for Dependency Preservation

- Compute F^+
- For each schema R_i in D do
- Begin
  - F_i:=restrictions of F^+ to R_i;
- End
- $F' := \Phi$
- For each restriction F_i do
- Begin
  - $F' = F' \cup F_i$
- End
- Compute F'^+
- If ($F'^+ = F^+$) then return true;
- else return false;

D is an input set and $D=\{R_1,R_2,\ldots,R_n\}$ of decomposed relation schemas

An example

Schema $R = (A, B, C)$

FDs $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

Decomposition $D = (R_1 = \{A, B\}, R_2 = \{B, C\})$

$\pi_{R_1}(F) = \{A \rightarrow B, B \rightarrow A\}$

$\pi_{R_2}(F) = \{B \rightarrow C, C \rightarrow B\}$

$$(\pi_{R_1}(F) \cup \pi_{R_2}(F))^+ = \{A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow B, A \rightarrow C, C \rightarrow A\} = F^+$$

Hence Dependency preserving

Algorithm for BCNF decomposition

R – given schema. F – given set of FDs

D = {R} // initial decomposition
while there is a relation schema R_i in D that is not in BCNF do
{ let $X \rightarrow A$ be the FD in R_i violating BCNF;
Replace R_i by $R_{i1} = R_i - \{A\}$ and $R_{i2} = X \cup \{A\}$ in D;
}

Decomposition of R_i is lossless as

$$R_{i1} \cap R_{i2} = X,\ R_{i2} - R_{i1} = A \text{ and } X \rightarrow A$$

Result: a lossless decomposition of R into BCNF relations

Dependencies may not be preserved (1/2)

S T D

Consider the schema: townInfo (stateName, townName, distName)
with the FDs F: $ST \rightarrow D$ (town names are unique within a state)
$D \rightarrow S$

Keys: ST, DT. – all attributes are prime
– relation in 3NF

Relation is not in BCNF as $D \rightarrow S$ and D is not a key

Decomposition given by algorithm: R_1: TD R_2: DS

Not dependency preserving as $\pi_{R_1}(F)$ = trivial dependencies
$\pi_{R_2}(F) = \{D \rightarrow S\}$

Union of these doesn't imply $ST \rightarrow D$

$ST \rightarrow D$ can't be enforced unless we perform a join.

Dependencies may not be preserved (2/2)

Consider the schema: R (A, B, C)

with the FDs F: $AB \rightarrow C$ and $C \rightarrow B$

Keys: AB, AC – relation in 3NF (all attributes are prime)

– Relation is not in BCNF as $C \rightarrow B$ and C is not a key

Decomposition given by algorithm: R_1: CB R_2: AC

Not dependency preserving as $\pi_{R_1}(F)$ = trivial dependencies

$\pi_{R_2}(F) = \{C \rightarrow B\}$

Union of these doesn't imply $AB \rightarrow C$

All possible decompositions: {AB, BC}, {BA, AC}, {AC, CB}

Only the last one is lossless!

Lossless and dependency-preserving decomposition doesn't exist.

Is Normalization Always Good?

- Example: Suppose A and B are always used together but normalization says they should be in different tables
    - Decomposition might produce unacceptable performance loss (always joining tables)
    - For example, data warehouses are huge historical DBs that are rarely updated after creation — joins are expensive or impractical
- Everyday DBs: aim for BCNF, settle for 3NF!

Database Design: Recap

- Closure algorithm to find keys
- Lossless decomposition
- 2NF
- 3NF
- BNCF