Relational Database Design(2)

Friday , March 6, 2015

REVIEW

- R= (ID, Name, City)
  - If ID is a key
- What are the FDs?

ID	Name	City
1	IBM	New York
2	MS	LA
3	Apple	Texas
4	MS	LA

REVIEW

- R= (ID, Name, City)
  - ID → Name

Which rows violate the FD?

(It take at least 2 rows to violate FD)

ID	Name	City
1	IBM	New York
1	MS	LA
3	Apple	Texas
4	MS	Seattle

REVIEW

- R= (ID, Name, City)
  - ID → Name

ID	Name	City
1	IBM	New York
1	MS	LA
3	Apple	Texas
4	MS	Seattle

Which rows violate the FD?

Row 1 and 2

ID→ Name

means

Each value of ID can has only a single name,

but no restriction on name.

REVIEW

- R= (ID, Name, City)
  - ID → Name,
  - ID → City,
  - Name → City

Which rows violate each FD?

ID	Name	City
1	IBM	New York
2	IBM	LA
3	Apple	Texas
4	MS	Seattle
4	MS	New York

(It take at least 2 rows to violate FD)

REVIEW

- R= (ID, Name, City)

  ID → Name,

  -

  ID → City,

  r4 and r5

  Name → City

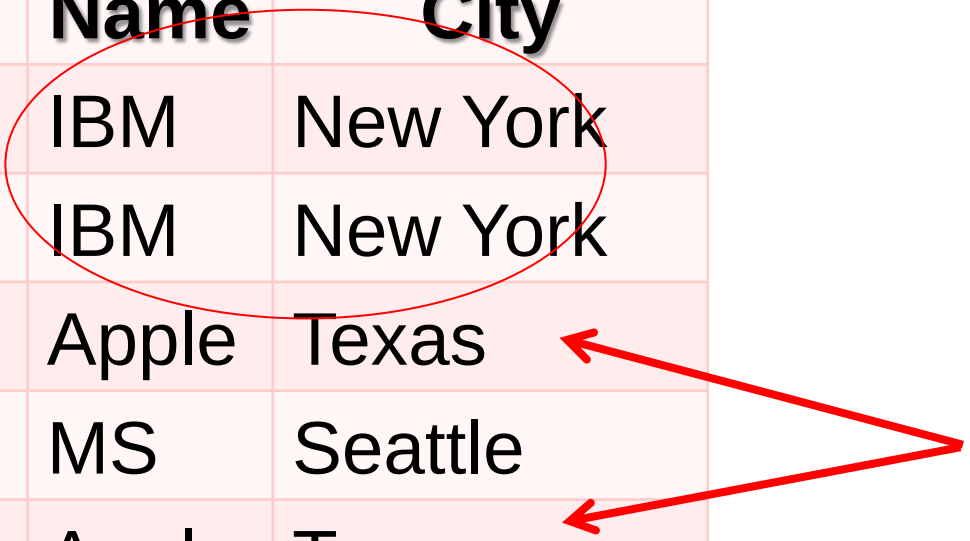
  r1 and r2, r4 and r5

ID	Name	City
1	IBM	New York
2	IBM	LA
3	Apple	Texas
4	MS	Seattle
4	MS	New York

Transitive Dependencies

- If ID → Name and Name → City
- then we have a new ID → City (transitive rule)

ID	Name	City
1	IBM	New York
2	IBM	New York
3	Apple	Texas
4	MS	Seattle
5	Apple	Texas



REVIEW

- R= (ID, Name, City)
  - ID → Name,
  - ID → City
  - Name → City
- What is a key?
- Is this a 2NF?
- Is this a 3NF?

ID	Name	City
1	IBM	New York
2	IBM	New York
3	Apple	Texas
4	MS	Seattle
5	Apple	Texas

Boyce-Codd Normal Form

1. *Domains of all attributes of R are atomic*
2. **Every determinant must be a superkey.**

2NF: all nonkey attributes are dependent on *the whole key*.

3NF: no transitive dependency.

Example

R = {ID, Name, city}

ID and Name are both keys.

ID → Name, City

Name → ID, City

- Is it in 3NF?

ID→Name and Name→City

- Any redundant data can exist?

No because Name → ID

ID	Name	City
1	IBM	New York
1	IBM	New York
2	MS	New York
3	Apple	Texas
4	HP	Seattle

Many-Many Relationship

Create an associative table !

SUPPLIER_PART

supplier_no	supplier_name	part_no	quantity
1	IBM	101	20
1	IBM	202	100
2	HP	101	50
2	HP	202	89
2	HP	303	70

What are the keys ?

Candidate Keys

- Assume that supplier names are always unique to each supplier.
- Thus we have two candidate keys:
  - **(supplier_no, part_no) and (supplier_name, part_no)**

supplier_no	supplier_name	part_no	quantity
1	IBM	101	20
1	IBM	202	100
2	HP	101	50
2	HP	202	89
2	HP	303	70

Functional Dependencies:

- Thus we have the following dependencies:
  - (supplier_no, part_no) -> quantity ✓
  - (supplier_name, part_no) -> quantity ✓
  - ***supplier_name*** *-> supplier_no* ***unique name**
  - ***supplier_no*** *-> supplier_name*
- No transitive dependency (so it is in 3NF)
- ***supplier_name*** → *supplier_no* → ***supplier_name***

Reflexive

Functional Dependencies:

- **Not BCNF ,** determinant is not a superkey.
  - ***supplier_name*** *-> supplier_no* ✗
  - ***supplier_no*** *-> supplier_name* ✗

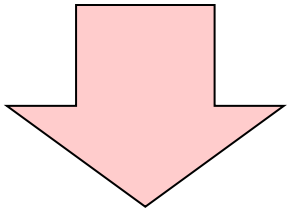
Duplicate Data Exist

supplier_no	supplier_name	part_no	quantity
1	IBM	101	20
1	IBM	202	100
2	HP	101	50
2	HP	202	89
2	HP	303	70

Decomposition (into BCNF):

SUPPLIER_PART

(supplier_no, supplier_name, part_no, quantity)



- Decompose the ***schema*** into:

SUPPLIER_ID (supplier_no, supplier_name)

SUPPLIER_PARTS (supplier_no, part_no, quantity)

Decompose 3NF into BCNF

SUPPLIER_PART (supplier_no, supplier_name, part_no, quantity)

SUPPLIER_ID

supplier_no	supplier_name
1	IBM
2	HP

SUPPLIER_PARTS

supplier_no	part_no	quantity
1	1	20
1	2	100
2	1	50
2	2	89
2	3	70

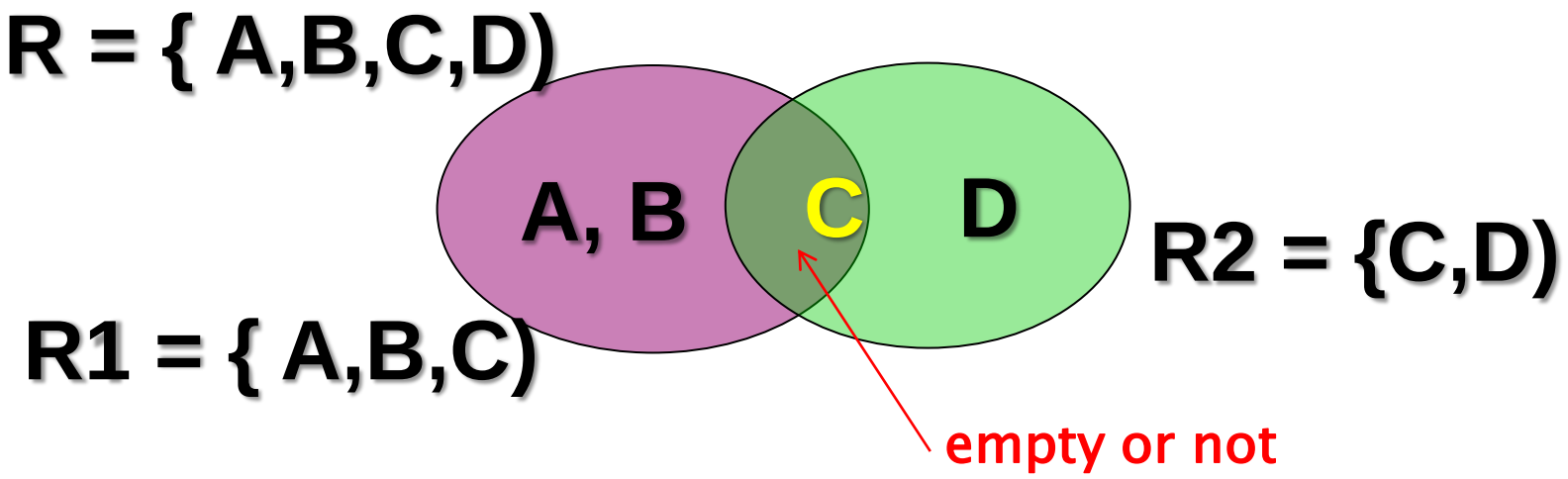
Goal — Devise a Theory for the Following

- **Decide** whether a particular relation ***R*** is in "good" form.
- In the case that a relation ***R*** is **not in "good" form, decompose** it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in a good form
  - the decomposition is a **lossless-join** decomposition
- Our theory is based on:
  - ***functional dependencies***
  - ***multi-valued dependencies***

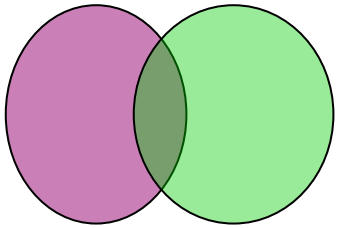
Decomposition

- All attributes of an original schema (*R)* must appear in the decomposition (*R*$_1$, *R*$_2$*):*

$$R = R_1 \cup R_2$$



Lossless–join Decomposition



- $R = R_1 \cup R_2$

- **Lossless-join** decomposition.
  For all possible relations *r* on schema *R*

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

Join give the same result as r

Π = **column projection**

$\bowtie$ = **cross product**

Functional Dependencies

Revisited

Functional Dependencies (Cont.)

- ***K*** is a **superkey** for relation schema ***R*** if and only if $K \rightarrow R$
- *K* is a candidate key for *R* iff
  - $K \rightarrow R$, and
  - no $Y \subset K$, $Y \rightarrow R$

minimal key

Example

R = (sid, name, grade)

sid,name → grade and

sid → name,grade

Hence {sid,name} is a super key.

but

{sid,name} is not a candidate key;

{sid} $\subset$ *{sid,name}* , sid $\rightarrow$ *R*

Functional Dependencies (Cont.)

- Functional dependencies allow us to express constraints that cannot be expressed using superkeys.

Use of Functional Dependencies

- Test relations legal under **F**.
  - ***whether*** r satisfies ***F.***
- Specify constraints on the set of legal relations
  - ***F*** holds on ***R*** if all legal relations on *R* satisfy *F.*

r1(R), r2(R) , r3(R) or tables r1, r2, r3 have the same schemas

Functional Dependencies (Cont.)

- **Trivial** functional dependency

  E.g.

  - **customer-name, loan-number → customer-name**
  - **customer-name → customer-name**

  - In general, **$Y \rightarrow X$ is trivial if $X \subseteq Y$**

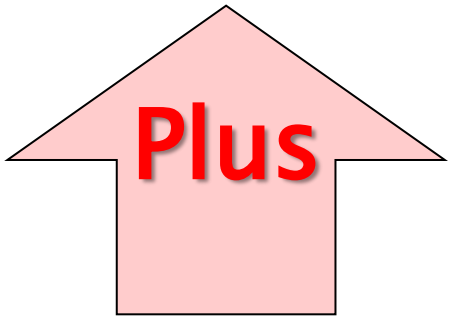
Closure of a Set of Functional Dependencies

- Given a set ***F*** set of functional dependencies, there are certain other functional dependencies that are *logically implied by* ***F***.
  - E.g. If **A** $\rightarrow$ **B** and **B** $\rightarrow$ **C**, then we can infer that ***A*** $\rightarrow$ ***C***
- The set of all functional dependencies logically implied by *F* is the ***closure of F*** ***(F^+)***

Example

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

$$F^{+} = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$$



Finding all of F^+

- Armstrong's Axioms:
  - if $X \subseteq Y$, then $Y \rightarrow X$ (reflexivity)
  - if $Y \rightarrow X$, then $ZY \rightarrow ZX$ (augmentation)
  - if $Y \rightarrow X$, and $X \rightarrow Z$, then $Y \rightarrow Z$ (transitivity)

Example

- **if $X \subseteq Y$, then $Y \rightarrow X$ (reflexivity)**

  trivial rule: ID, Name → ID ;always true

  ID → ID

- **if $Y \rightarrow X$, then $ZY \rightarrow ZX$ (augmentation)**

  If ID → GPA then

  ID,*Name* → GPA,*Name*

Example (transitivity rule)

- If Y $\rightarrow$ *X*, and *X* $\rightarrow$ Z, then Y $\rightarrow$ Z

IF ID → Name, Name → City

THEN *ID → City*

Homework

- Consider 1NF, 2NF, 3NF, BCNF
- Compute F+ according to the Armstrong axioms

Example

- $R = (A, B, C, G, H, I)$

  $F = \{ A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H \}$

- some members of F^+
  - $A \rightarrow H$
  - We have $A \rightarrow B$, $A \rightarrow C$, $CG \rightarrow H$, $B \rightarrow H$
    - Can we derive from $A \rightarrow C$, $CG \rightarrow H$,
      - NO

    contain only single A on left hand side

    cannot derive
  - We have $A \rightarrow B$ and $B \rightarrow H$
    - we can derive $A \rightarrow H$ (transitivity)

Example

- ***R = (A, B, C, G, H, I)***
  F = { A→B, A→C, CG→H, CG→I, B→H }
- ***AG → I***
  - We can eliminate A-> B, B->H, ***CG→H***
    - No **H** on LHS of any FD
  - **Now consider *A→C, CG→I,***
    1. ***AG → CG*** **(by augmenting *A → C* with G)**
    2. ***CG → I*** ***(from F)***
    3. ***AG → I*** ***(1-2 and* transitivity)**

Union Rule

***F* = {**ID→ name , ID→ dept **}**

- ***ID → Name, Dept***
  - ❖from ***ID→ Name and ID → Dept :*** **"union rule"** can be inferred from FD definition,
    1. ***ID → IDName*** *by augmenting ID→Name with ID*
    2. ***ID Name→Dept Name*** *by* augmenting *ID→Dept with Name,*
    3. ***ID → Dept Name*** ***1-2*** transitivity

Exercise

$$F = \{ A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H \}$$

- ***Derive CG → HI using Axiom rules***
  - from ***CG → H and CG → I :*** "**union rule**" can be inferred from FD definition, **or**

Exercise

$F = \{ A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H \}$

- ***Derive CG → HI by***
- *Consider CG → H, CG → I,*
  - **CG → CGI** by augmenting **CG → I** *with* **CG**
  - **CGI → HI** *by* augmenting **CG → H** *with* **I**,
  - **CG → HI** transitivity

It is
{C,G} → {H,I}

Procedure for Computing F^+

- To compute the closure of a set of FD F:
- $F^+ = F$

  repeat

    for each FD f in F^+

      apply **reflexivity and augmentation** rules on f

      add the resulting FD to F^+

    for each pair of FD f_1 and f_2 in F^+

      if f_1 and f_2 can be combined **using transitivity**

      then add the resulting FD to F^+

  until F^+ does not change any further

There exists an easier method

Closure of Functional Dependencies (Cont.)

- We can further simplify manual computation of F^+ by using the following additional rules.
  - **If** $Y \rightarrow X$ and $Y \rightarrow Z$ holds, **then** $Y \rightarrow XZ$ holds **(union)**
  - **If** $Y \rightarrow XZ$ holds, **then** $Y \rightarrow X$ and $Y \rightarrow Z$ holds **(decomposition)**
  - **If** $Y \rightarrow X$ and $ZX \rightarrow W$ holds, **then** $ZY \rightarrow W$ holds **(pseudotransitivity)** *replace x with y*

  The above rules can be inferred from Armstrong's axioms.

Closure of Attribute Sets

Given a set of attributes ***G*** ,

Define the *closure* of ***G*** under *F* (***G*$^+$**) as the set of attributes that are functionally determined by ***G*** under ***F:***

$$\mathbf{G \rightarrow X} \text{ is in } \boldsymbol{F^+} \Leftrightarrow X \subseteq \boldsymbol{G^+}$$

G can determine X, if X is a subset of G^+

Closure of Attribute Sets

- Algorithm to compute G^+, the closure of G under F

```
result := G;
while (changes to result) do
for each X → Y in F do
begin
    if X ⊆ result then result := result ∪ Y
end
```

G^+ = Result

Example of Attribute Set Closure

R = (A,B,C,G,H,I)

F = { A → B, A → C,CG → H, CG → I, B → H }

$(AG)^+$

1. ***result = AG*** *start with AG*
2. ***result = ABCG*** *add BC (A → C, A → B ; A ⊆ result)*
3. ***result = ABCGH*** *add H (CG → H; CG ⊆ result)*
4. ***result = ABCGHI*** *add I (CG → I; CG ⊆ result)*

$(AG)^+ = ABCGHI$

R = (A,B,C,G,H,I)

F = { $A \rightarrow B$, $A \rightarrow C$, $CG \rightarrow H$, $CG \rightarrow I$, $B \rightarrow H$ }

Is *AG* a candidate key?

1. Is AG a super key?

1. Does $AG \rightarrow R$? == Is $(AG)^+ \supseteq R$ yes

2. Is any subset of AG a superkey?

1. Does $A \rightarrow R$? == Is $(A)^+ \supseteq R$ no
2. Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$ no

So *AG* is a candidate key

Uses of Attribute Closure

- **Testing for a superkey:**
  - compute Y^+, and check if $Y^+ \supseteq R$.
- **Testing functional dependencies**
  - To check if $Y \rightarrow X$ holds (or, is in F^+),
    - just check if $X \subseteq Y^+$.
  - a simple and cheap test, and very useful

Ex. $(AG)^+ = ABCGHI$

$AG \rightarrow B$, $AG \rightarrow H$,…

Uses of Attribute Closure

- ***Computing closure of F***
  - **For each $A \subseteq R$, we find the closure A^+, and**
  - **For each $S \subseteq A^+$** ($A^+ = \{ S, \ldots. \}$)
    - **we output a functional dependency**

$$A \rightarrow S.$$

Computing closure of F

- R = (A,B,C,G,H)
- F = { A → B,
  B → C,
  CG → H }

For each $X \subseteq R$, we find the closure X^+,

A^+ = {A,B,C}
B^+ = { B,C }
C^+ = { C }
G^+ = { G }
H^+ = { H }
CG^+ = {C,G,H }
AG^+ = {A,B,C,G,H } = R
BG^+ = {B,C,G,H }

No need to compute AB^+, BC^+, AC^+ (trivial)

For each $S \subseteq X^+$, output $X \rightarrow S$.

- F^+ = { A → B, A → C,
  CG → H , B → C,
  AG → H, AG → B,
  AG → C, BG → H,
  BG → C }

Benefit of F^+

Normalization

Goals of Normalization

- Decide whether a particular relation *R* is in "good" form.
- In the case that a relation *R* is not in "good" form, decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies

Decomposition

- All attributes of an original schema (*R*) must appear in the decomposition (R_1, R_2):

$$R = R_1 \cup R_2$$

- Lossless-join decomposition.
  For all possible relations *r* on schema *R*

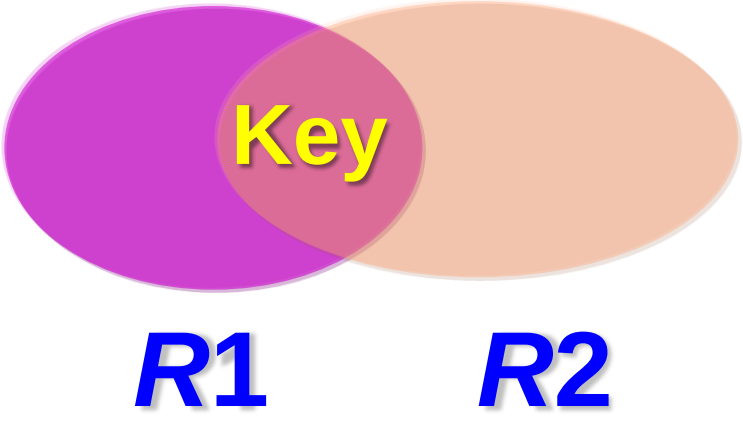
$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

Result of join must = r

Lossless-join Decomposition

- A decomposition of **R into R_1 and R_2** is lossless join if and only if at least one of the following dependencies is in F^+:
  - $(R_1 \cap R_2) \rightarrow R_1$ **or**
  - $(R_1 \cap R_2) \rightarrow R_2$

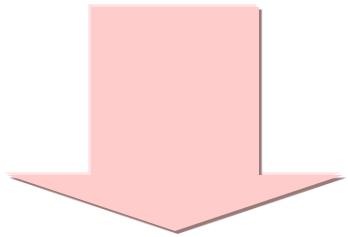
Key for joining R_1, R_2



Decomposition

Employee

Name	Address	Province	Phone1	Phone2
John	1	01	-	-
John	2	01	021111111	022222222
Mary	3	02	-	025555555
Sam	4	02	-	-



Decomposition

Employee(R1)

Name	Address	Province
John	1	01
John	2	01
Mary	3	02
Sam	4	02

R1 = { Name, Address, Province}
R2 = { Name, Phone }

R1 $\cap$ R2 = { Name }

Name $\twoheadrightarrow R_1$, Name $\twoheadrightarrow R_2$

Telephone(R2)

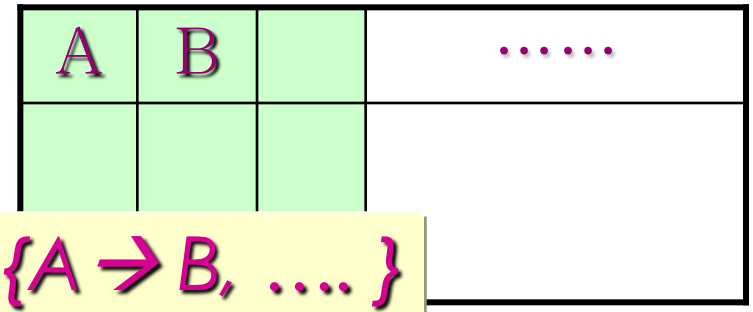
Name	Phone
John	021111111
Mary	025555555
John	022222222

Normalization Using Functional Dependencies

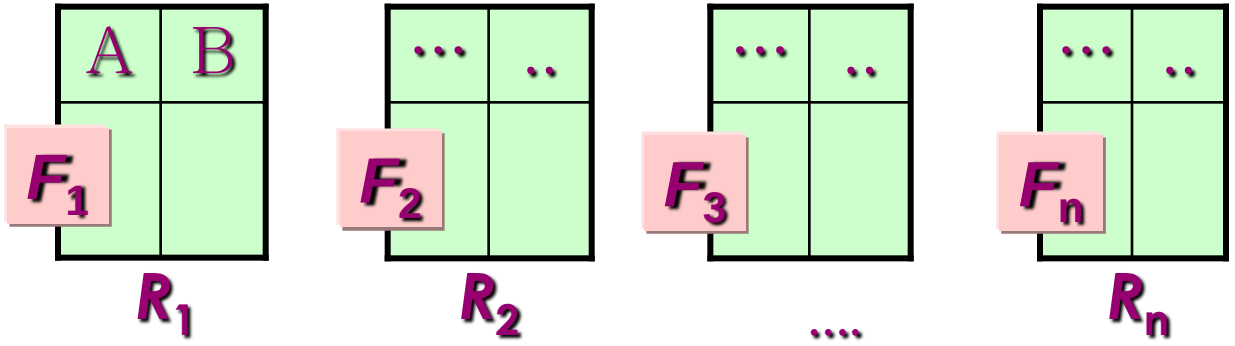
- When we decompose a relation schema R with a set of functional dependencies F into R_1, R_2,.., R_n we want
  - Lossless-join decomposition: Otherwise information loss.
  - No redundancy: R_i should be in either Boyce-Codd Normal Form or Third Normal Form.
  - Dependency preservation: the decomposition should be dependency preserving

Dependency preservation

- *Before*



- *After decompose R into* $R_1, R_2, .., R_n$



- **dependency preserving**

$$(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$$

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
  - Can be decomposed in two different ways

R

A	B	C

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

- *First Method*

A	B

R_1

B	C

R_2

$F_2 = \{B \rightarrow C\}$

$F_1 = \{A \rightarrow B\}$

- Lossless-join decomposition:

  $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$

- **Dependency preserving**

  $(F_1 \cup F_2)^+ = F^+$

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
  - Can be decomposed in two different ways

R

A	B	C

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

$F^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$

- ***Second Method***

A	B

R_1

A	C

R_2

$F_2 = \{A \rightarrow C\}$

$F_1 = \{A \rightarrow B\}$

- Lossless-join decomposition:

$R_1 \cap R_2 = \{A\}$ and $A \rightarrow B$

- **No dependency preserving**

$(F_1 \cup F_2)^+ \neq F^+$

(cannot check $B \rightarrow C$ without computing $R_1 \times R_2$)

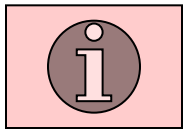
Boyce-Codd Normal Form

A relation schema *R* is in BCNF with respect to a set *F* of FD if for all FDs in F^+ of the form $Y \rightarrow X$,
where $Y \subseteq R$ and $X \subseteq R$,

at least one of the following holds:

- $Y \rightarrow X$ **is trivial (i.e., $X \subseteq Y$) หรือ**
- **Y is a superkey for *R***

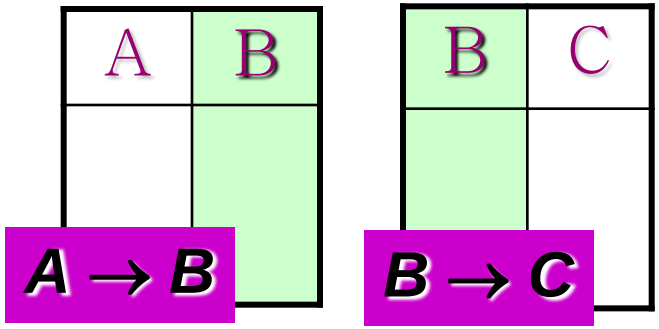
  all determinants are keys.



BCNF

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
  Key = $\{A\}$
- R is not in BCNF (B is not a key but $B \rightarrow C$)
- Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$
  - R_1 and R_2 in BCNF
  - Lossless-join decomposition
  - Dependency preserving



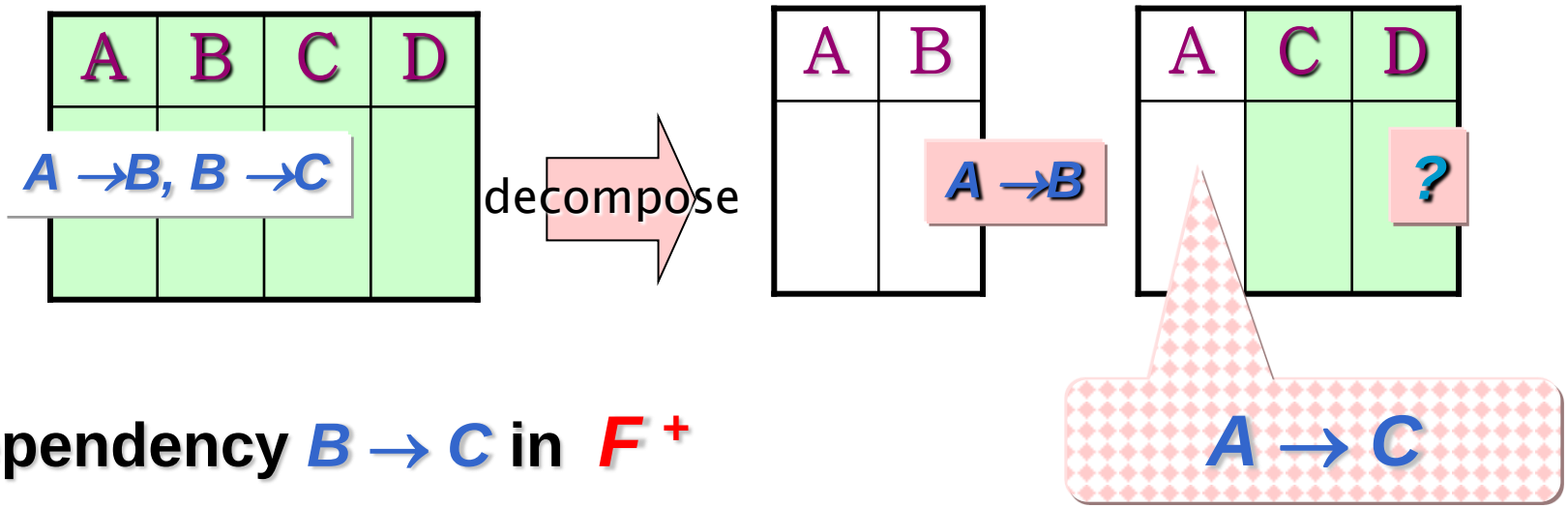
Testing for BCNF

- To check if a non-trivial dependency $X \rightarrow Y$ causes a violation of BCNF
  1. **compute X^+ and**
  2. **verify that X^+ includes all attributes of R, that is, X is a superkey of R.**

- check **only** the dependencies in the given set F for violation of BCNF, rather than checking all dependencies in F^+.

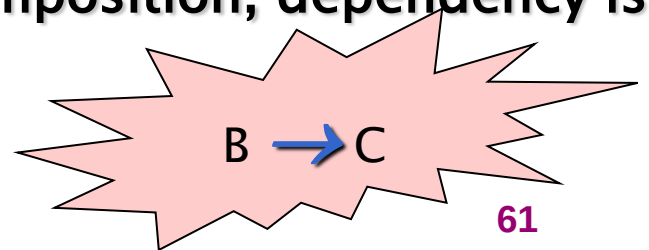
Testing for BCNF

- However, using only F is **incorrect** when testing a relation in a **decomposition** of **R**
  - E.g. Consider ***R* (*A, B, C, D*)**, with ***F* = { *A* →*B*, *B* →*C* }**
    - Decompose *R* into $R_1(A,B)$ and $R_2(A,C,D)$



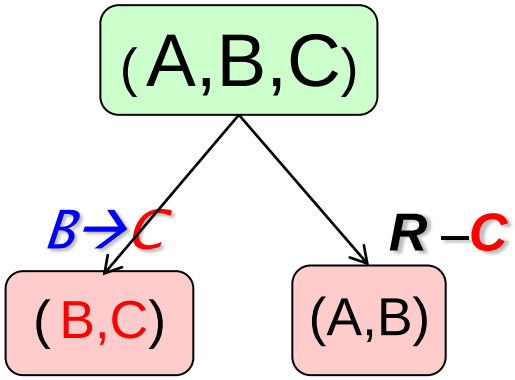
- **dependency $B \rightarrow C$ in F^+**

After decomposition, dependency is not preserved



BCNF Decomposition Algorithm

- **$R = (A, B, C)$ $F = \{A \rightarrow B, B \rightarrow C\}$**
  Key = {*A*}
- **R is not in *BCNF***
  (B is not a key but $B \rightarrow C$)



Decomposition

- ***Result* = { R };** *Compute F+={A→B,B →C,A →C }*
  - ***Repeat***
    - ***Select** Ri =(ABC)* **not in BCNF***
      - ***Select** B →C in F+*
      - *Remove Ri from the result*
      - ***Result =** { R_1=(B,C), R_2= Ri-C=(A,B) }*
  - ***Until done** (No Ri not in BCNF)*

BCNF Decomposition Algorithm

result := {*R* };
done := false;
compute F^+;

Start with R not in BCNF

F^+ on R

Check for FD that X is not a key

```
while (not done) do
   if (there is a schema Ri in result that is not in BCNF)
   then
      begin
            let  X → Y   be a  nontrivial FD on Ri
            result := (result – Ri ) ∪ {X, Y } ∪ (Ri – Y);
      end
   else done := true;
```

1.Remove R_i to decompose

2. Add R1
Add a schema in BCNF

3.Add R2: the rest of R_i without Y, keep X for lossless join

Note: each R_i is in BCNF, and decomposition is lossless-join.

Example of BCNF Decomposition

- $R = (A , B, C, D, E, F)$

  $F = \{A \rightarrow CB, E \rightarrow FA \}$

  Key = *{E, D}*

Not in BCNF because
A nor E is not a key

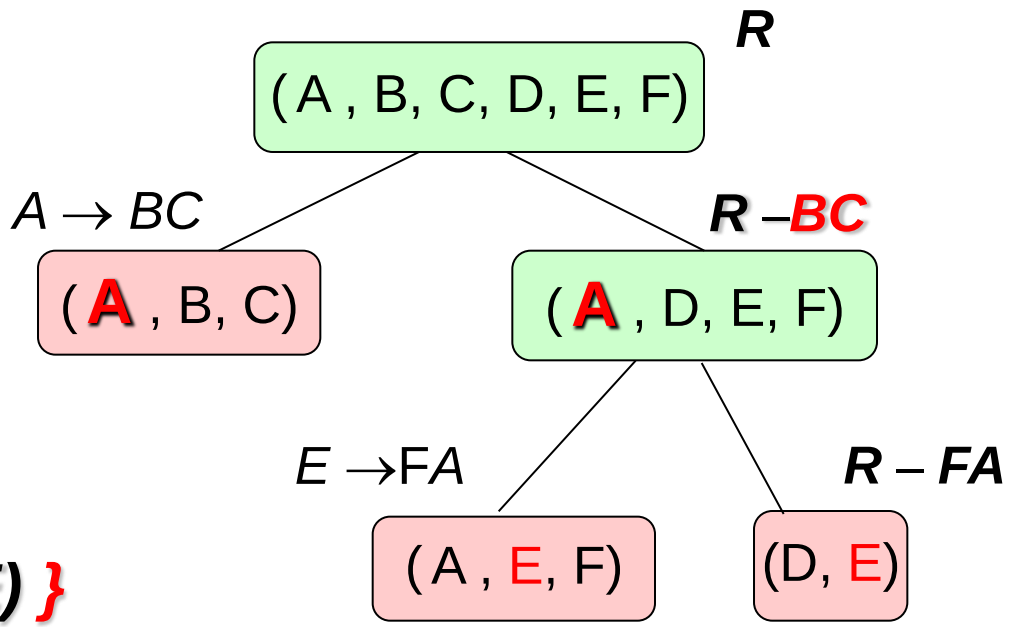
- Need Decomposition
- First *compute F+ by*
- **For each $A \subseteq R$, we find the closure A^+**
  - $A^+ \rightarrow ACB$
  - $E^+ \rightarrow FACB$
  - $(ED)^+ \rightarrow FABCD$
- $F+ = \{A \rightarrow CB, E \rightarrow ACBF, ED \rightarrow ABCDF\}$

Example of BCNF Decomposition

- $R = (A, B, C, D, E, F)$

  $F+ = \{A \rightarrow BC,\ E \rightarrow ABCF,\ ED \rightarrow ABCDF\}$

- Decomposition



Final decomposition

{ (A,B,C), (A,E,F), (D,E) }

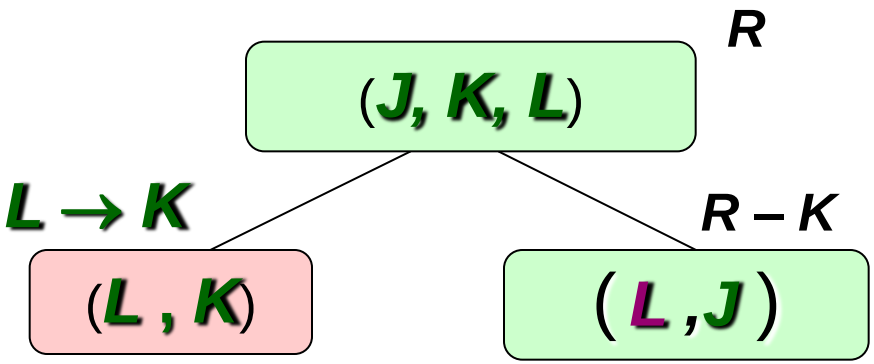
(F1 U F2 U F3)+ equals to F+

All the leave nodes in the result

BCNF and Dependency Preservation

It is not always possible to get a BCNF decomposition that is dependency preserving

- $R = (J, K, L)$
  $F = \{JK \rightarrow L, L \rightarrow K\}$
  Two candidate keys = *JK* and *JL*
- ***R* is not in BCNF (∈ L-->K but L is not a key)**
- **Any decomposition of *R* will fail to preserve $JK \rightarrow L$**



Question ?