Functional Dependencies 2

Wk8 Wed

Closure of a set of FDs

- The set of all FDs implied by a given set F of FDs is called the closure of F, denoted as F^+.
- Armstrong's Axioms, can be applied repeatedly to infer all FDs implied by a set of FDs.

Suppose X,Y, and Z are sets of attributes over a relation.

Armstrong's Axioms

- Reflexivity: if $Y \subseteq X$, then $X \rightarrow Y$ *(trivial dependency)*
- Augmentation: if $X \rightarrow Y$, then $XZ \rightarrow YZ$
- Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

student_ID	student_name	course_ID	course_name	department_name
111	Chan	3170	DB	CSE
222	Wong	3170	DB	CSE
333	Tam	3160	Cal	MATH
111	Chan	3160	Cal	MATH

reflexivity:

student_ID, student_name → student_ID

student_ID, student_name → student_name

augmentation:

student_ID → student_name
implies
student_ID, course_name → student_name, course_name

transitivity:

course_ID → course_name and course_name →department_name

Implies course_ID → department_name

Recall: Closure of a set of FDs

- Armstrong's Axioms is sound and complete.
  - Sound: they generate only FDs in F^+.
  - Complete: repeated application of these rules will generate all FDs in F^+.

Armstrong's Axioms (Cont.)

- Additional Rules we inferred from Armstrong's axioms.
  - Rule 4 (**additivity**):
    - If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds
  - Rule 5 (**projectivity**):
    - If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds
  - Rule 6 (**pseudo-transitivity**):
    - If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds

- **Note: Other names:**
  Additivity aka Union
  Projectivity aka Decomposition

Symbol **|=** denotes infers, i.e., FDs on LHS infer the FDs on RHS

Armstrong's Axioms Rule 5

Proving rule 5: *projectivity*

$\{X \rightarrow Y Z\} \models X \rightarrow Y$

Cheat Sheet

F1 (Reflexivity) If $X \supseteq Y$ then $X \rightarrow Y$.

F2 (Augmentation) $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.

F3 (Transitivity) $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.

Armstrong's Axioms Rule 5

To show correctness of the projectivity rule:

if $X \to YZ$, then $X \to Y$ *(*and $X \to Z$*)* (**projectivity**)

Proof:

$X \to YZ$ … (1) (given)

$YZ \to Y$ … (2) (reflexivity)

$X \to Y$ … (3) (transitivity on (1), (2))

$YZ \to Z$ … (4) (reflexivity)

$X \to Z$ … (5) (transitivity on (1), (4))

Armstrong's Axioms Rule 6

Proving rule 6: *Pseudo-transitivity*

$\{X \rightarrow Y, YZ \rightarrow W\} \models XZ \rightarrow W$

Cheat Sheet

F1 (Reflexivity) If $X \supseteq Y$ then $X \rightarrow Y$.

F2 (Augmentation) $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.

F3 (Transitivity) $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.

Armstrong's Axioms Rule 4

Proving rule 4: *Additivity*

$\{X \to Y, X \to Z\} \models X \to Y Z$

Cheat Sheet

F1 (Reflexivity) If $X \supseteq Y$ then $X \to Y$.

F2 (Augmentation) $\{X \to Y\} \models XZ \to YZ$.

F3 (Transitivity) $\{X \to Y, Y \to Z\} \models X \to Z$.

Solution

To show correctness of the *Additivity* rule:

$X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$ (***Additivity***)

Proof:

$X \rightarrow Y$	… (1) (given)
$X \rightarrow Z$	… (2) (given)
$XX \rightarrow XY$	… (3) (augmentation on (1))
$X \rightarrow XY$	… (4) (simplify (3))
$XY \rightarrow ZY$	… (5) (augmentation on (2))
$X \rightarrow ZY$	… (6) (transitivity on (4) and (5))

FD Inference - Practice

Cheat Sheet

F1 (Reflexivity) If $X \supseteq Y$ then $X \rightarrow Y$

F2 (Augmentation) $\{X \rightarrow Y\} \models XZ \rightarrow YZ$

F3 (Transitivity) $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$

F4 (Additivity) $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$

F5 (Projectivity) $\{X \rightarrow YZ\} \models X \rightarrow Y$

F6 (Pseudo-transitivity) $\{X \rightarrow Y, YZ \rightarrow W\} \models XZ \rightarrow W$

Given F = {A → B, A → C, BC → D}

Prove A → D:

Recall: F and it's Closure

- Definition. the set of all dependencies that can be inferred from F is called the **closure** of F
  - F^+ denotes the closure of F
  - F^+ includes dependencies in F
- Note:
  - We typically reserve F to denote the set of functional dependencies that are specified on relation schema R.

Key Points on Closures

1. F denotes the set of FD’s of a relation
2. F^+ is the **closure** of F
3. F^+ is the set of FD’s that
   - F^+ includes dependencies in F
   - F^+ is **closed** under Armstrong’s axioms
     - Closure Example “A set is closed under addition if you can add any two numbers in the set and still have a number in the set as a result.”

- How do we check if a functional dependency can be inferred from FD’s F (is a member of F^+)?

Procedure for Computing F^+

- To compute the closure of a set of functional dependencies F:

$F^+ = F$
repeat
for each functional dependency f in F^+
apply reflexivity and augmentation rules on f
add the resulting functional dependencies to F^+
for each pair of functional dependencies f_1 and f_2 in F^+
if f_1 and f_2 can be combined using transitivity
then add the resulting functional dependency to F^+
until F^+ does not change any further

$R \quad = (A, B, C)$

$F \quad = \{ \quad A \to B, B \to C \quad \}$

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

$AB \to AB, \; BC \to BC, \; AC \to AC, \; ABC \to ABC,$

$AB \to A, \; AB \to B,$

$BC \to B, \; BC \to C,$

$AC \to A, \; AC \to C,$

$ABC \to AB, ABC \to BC, ABC \to AC,$

$ABC \to A, ABC \to B, ABC \to C,$

Using reflexivity, we can generate all trivial dependencies

$A \to B,$ … (1) (given)

$B \to C,$ … (2) (given)

$A \to C,$ … (3) (transitivity on (1) and (2))

$AC \to BC,$ … (4) (augmentation on (1))

$AC \to B,$ … (5) (decomposition on (4))

$A \to AB,$ … (6) (augmentation on (1))

$AB \to AC, \; AB \to C, B \to BC,$

$A \to AC, \; AB \to BC, AB \to ABC, \; AC \to ABC, A \to BC, A \to ABC \}$

A Motivating Example

- Given F = { X → Y, Y → Z}
- Possible Question: Can X → Z be inferred or derived from the FDs in F?
  - Should we check for X → Z through checking membership in F+ (by computing F+) ?
- If so…
  - F+ = {XY → X, XY → Y, XY → Z, XZ → X, XZ → Y, XZ → Z, XYZ → X, XYZ → Y, XYZ → Z, XY → XY, XY → YZ, XY → XZ, … , X → Z , … }
  - Based on F+, X → Z is in the closure of F.

Attribute Closure

- Computing the closure of a set of FDs can be expensive
- In many cases, we just want to check if a given FD
  $X \rightarrow Y$ is in F^+.

- So for when checking for $X \rightarrow Z$, given F = { $X \rightarrow Y, Y \rightarrow Z$}
  1. Compute X^+ instead of F^+
  2. We then check if Z is covered by X^+

  Where X and Z - a set of attributes

  Where F - a set of functional dependencies

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

Example

$$
\begin{aligned}
F &= \{ A \rightarrow B, B \rightarrow C \} \\
A^+ &= ABC \\
B^+ &= BC \\
C^+ &= C \\
AB^+ &= ABC
\end{aligned}
$$

Computing Attribute Closures

Pseudocode to the closure of A under *F*

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

When no additional changes to result is possible, the final value of variable result is A^+

Attribute Closures – Practice A

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

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

Task: Compute the closure of AG

Cheat Sheet:

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

Attribute Closures – Solution A

To compute AG^+

$result = AG$

$result = ABG \quad (A \rightarrow B)$

$result = ABCG \quad (A \rightarrow C)$

$result = ABCGH \quad (CG \rightarrow H)$

$result = ABCGHI \quad (CG \rightarrow I)$

Computing Attribute Closures

The equivalent algorithm, should you implement it

```
X := X;
change := true;
while change do
        begin
        change := false;
        for each FD W → Z in F do
                begin
                if (W ⊆ X+) and (Z ⊈ X+) then do
                        begin
                        X+ := X+ ∪ Z;
                        change := true;
                        end
                end
        end
```

Try it yourself: Exercise

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

Practice: **Compute A+**

Cheat Sheet:

```
X+ := X;
change := true;
while change do
        begin
        change := false;
        for each FD W → Z in F do
          begin
          if (W ⊆ X+) and (Z ⊈ X+)
          then do
                begin
                X+ := X+ ∪ Z;
                change := true;
                end
          end
        end
```

Part II

FDS AND KEYS

Recall Exp. of Attribute Set Closure

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

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

We know $(AG)^+ = ABCGHI$

Observation: could AG a candidate key?

Is AG a super key?

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

Is any subset of AG a super key?

Does $A \rightarrow R$? == Is $(A)^+ \supseteq R$

Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$

Note: γ and α each refer to a set of attributes

Functional Dependencies

- α is a super key for R iff $\alpha \rightarrow R$ where R is schema for a relation R.

- α is a candidate key for R iff
  - $\alpha \rightarrow R$, and
  - for no γ that is a proper subset of α, $\gamma \rightarrow R$ (minimal property).

Functional Dependencies

Assuming...

student_ID → student_name

course_ID → course_name

student_ID	student_name	course_ID	course_name
111	Chan Tai Man	3170	Database
222	Wong Siu Ling	3170	Database
333	Tam Wai Ming	3160	Algorithms
111	Chan Tai Man	3160	Algorithms

- (student_ID, course_ID) is a candidate key
- (student_ID, course_ID, course_name) is not a candidate key

Example

- Consider schema *STUDENT(zid, name, address)*
- Where *zid* $\rightarrow$ *name, address*

- Notes:
  - Key of a relation will always functionally determine every attributes in the relation
  - Left-hand side of a dependency does not imply uniqueness

Functional Dependencies

- Functionally dependencies are a generalization of a concept of a key

- Consider the schema:

  inst_dept (<u>*ID, dept_name,*</u> *name, salary, building, budget*)

- We can also express functional dependencies to hold:

$$dept\_name \rightarrow building$$

$$ID \rightarrow building$$

Procedurally Determine Keys

- Motivation: use FDs to procedurally determine the keys of a relation.

Procedurally Determine Keys

- How to compute a candidate key of a relation R
  based on the FD's belonging to R
- Algorithm:
  - *Step 1 : Assign a super-key of R in F to X.*
  - *Step 2 : Iteratively remove attributes from X while retaining the property X+ = R till no reduction on X is possible.*
  - *The remaining X is a key.*
- Let's try an example

Practice

Step 1 : Assign a super-key of R in F to X.
Step 2 : Iteratively remove attributes from X while retaining the property $X^+ = R$ till no reduction on X is possible.
The remaining X is a key.

Given:

$R = \{A, B, C, D\}$

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

Compute all Candidate Keys

- Given a relational schema R and a set of functional dependencies F on R, find all the possible ways we can identify a row.
- Note: we know how to compute one candidate key already.

Compute all Candidate Keys

- The algorithm to compute all the candidate keys is as follows:

```
T := Ø
Main:
        X := S
        remove := true
        While remove do
                For each attribute A ∈ X
                Compute {X-A}+ with respect to F
                If {X-A}+ contains all attributes of R then
                        X := X – {A}
                Else
                        remove := false
        T :=T ∪ X
```

- Repeat until no available S can be found.
  Finally, T contains all the candidate keys.

Compute all Candidate Keys

- Given relation R(A, B, C, D, E)
- R contains a set of FDs {A → B, BC → A, D → E}
- **Task:** Find all the candidate keys for relation R

Compute all Candidate Keys

```
T := ∅
Main:
        X := S
        remove := true
        While remove do
            For each attribute A ∈ X
            Compute {X-A}+ with respect to F
            If {X-A}+ contains all attributes of R then
                  X := X – {A}
            Else
                  remove := false
        T :=T ∪ X
```

Relation R(A, B, C, D, E)
with set of FDs {A → B, BC → A, D → E}

Compute all Candidate Keys

```
T := Ø
Main:
        X := S
        remove := true
        While remove do
            For each attribute A ∈ X
            Compute {X-A}+ with respect to F
            If {X-A}+ contains all attributes of R then
                  X := X – {A}
            Else
                  remove := false
        T :=T ∪ X
```

Relation R(A, B, C, D, E)
with set of FDs {A → B, BC → A, D → E}

Compute all Candidate Keys

```
T := Ø
Main:
        X := S
        remove := true
        While remove do
            For each attribute A ∈ X
            Compute {X-A}+ with respect to F
            If {X-A}+ contains all attributes of R then
                  X := X – {A}
            Else
                  remove := false
        T :=T ∪ X
```

Relation R(A, B, C, D, E)
with set of FDs {A → B, BC → A, D → E}

Lecture Learning Outcomes

- Take aways
  - Functional Dependencies
  - Armstrong’s axioms
  - Given a FD, check if the FD can be derived from a given set of FD
  - How to compute one candidate key
  - How to compute all candidate keys

- Please Revisit Lecture on Normal Forms in your own time

Acknowledgement: Several slides in this lecture were inspired by the textbook slides made by Siberschatz, Korth and Sudarshan from Database System Concepts. 6th Ed.

Normal Forms

1st Normal Form	No repeating data groups
2nd Normal Form	No partial key dependency
3rd Normal Form	No transitive dependency
Boyce-Codd Normal Form	Reduce keys dependency
4th Normal Form	No multi-valued dependency
5th Normal Form	No join dependency

$$1NF \supset 2NF \supset 3NF \supset BCNF \supset 4NF \supset 5NF$$

Decomposition

- Decomposition is a tool that allows us to eliminate redundancy.
- It is important to check that a decomposition does not introduce new problems.
  - A decomposition allows us to recover the original relation?
  - Can we check integrity constraints efficiently?

Decomposition

A set of relation schemas { R_1, R_2, …, R_n }, with $n \geq 2$ is a **decomposition** of R if $R_1 \cup R_2 \cup \ldots \cup R_n = R$

Supply

sid	status	city	part_id	qty

Supplier

sid	status	city

and

SP

sid	part_id	qty

Problems with decomposition

1. Some queries become more expensive.
2. Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation – information loss.
3. Checking some dependencies may require joining the instances of the decomposed relations.

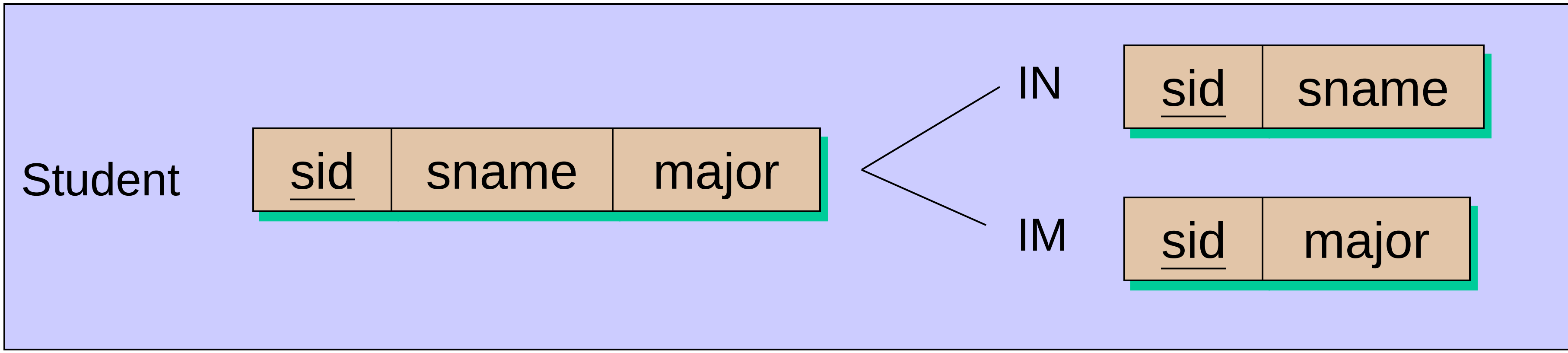
Lossless Join Decomposition

The relation schemas { R_1, R_2, …, R_n } is a **lossless-join decomposition** of R if

For all possible relation instances *r* on schema *R*,

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

Example: a lossless join decomposition



Student

sid	sname	major
123	Ling Wang	C.S.
456	Hong Zhou	C.S.

IN

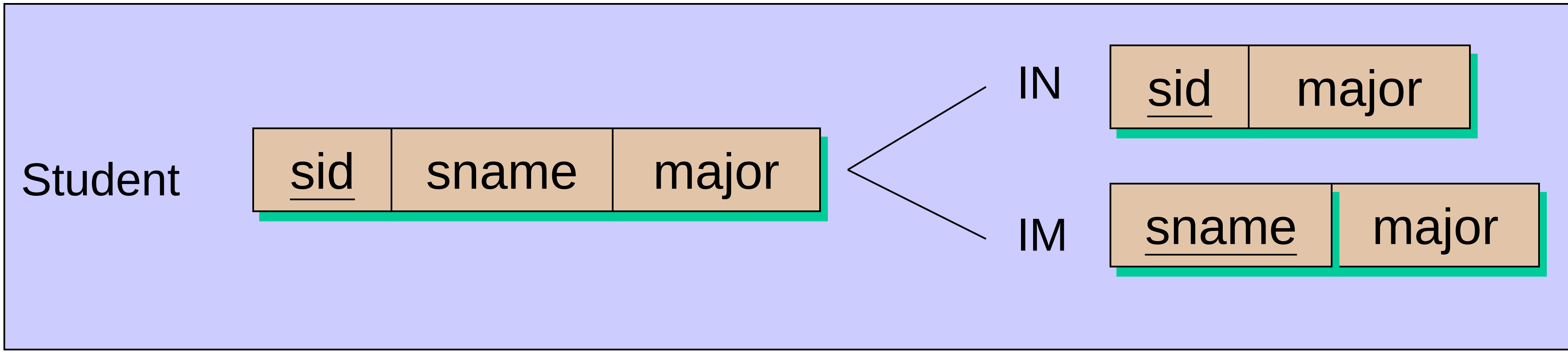
sid	sname
123	Ling Wang
456	Hong Zhou

IM

sid	major
123	C.S.
456	C.S.

'Student' can be recovered by joining the instances of IN and IM

Example: a non-lossless join decomposition



Student

sid	sname	major
123	Ling Wang	C.S.
456	Hong Zhou	C.S.

IN

sid	major
123	C.S.
456	C.S.

IM

sname	major
Ling Wang	C.S.
Hong Zhou	C.S.

Student = IN $\bowtie$ IM????

IN

sid	major
123	C.S.
456	C.S.

IM

sname	major
Ling Wang	C.S.
Hong Zhou	C.S.

$\text{IN} \bowtie \text{IM}$

sid	sname	major
123	Ling Wang	C.S.
123	Hong Zhou	C.S.
456	Ling Wang	C.S.
456	Hong Zhou	C.S.

$\neq$

Student

sid	sname	major
123	Ling Wang	C.S.
456	Hong Zhou	C.S.

The instance of ‘Student’ cannot be recovered by joining the instances of IN and IM. Therefore, such a decomposition is not a lossless join decomposition.

Theorem:

R - a relation schema

F - set of functional dependencies on R

The decomposition of R into relations with attribute sets

$\boldsymbol{R_1}$**,** $\boldsymbol{R_2}$ **is a lossless-join decomposition** iff

$$(R_1 \cap R_2) \rightarrow R_1 \in F^+$$

OR

$$(R_1 \cap R_2) \rightarrow R_2 \in F^+$$

i.e., $R_1 \cap R_2$ is a **superkey** for R_1 or R_2.

(the attributes common to R_1 and R_2 must contain a key for either R_1 or R_2).

- Example
  - R = (A, B, C)
  - F = { A $\longrightarrow$ B }
  - R = { A, B } + { A, C } is a lossless join decomposition
  - R = { A, B } + { B, C } is not a lossless join decomposition

- See if you can relate this to the previous relation 'Student'

Another Example

$R = \{ A, B, C, D \}$

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

Decomposition: $\{ (A, B), (C, D), (A, C) \}$

Consider it a two step decomposition:

1. Decompose R into $R_1 = (A, B)$, $R_2 = (A, C, D)$
2. Decompose R_2 into $R_3 = (C, D)$, $R_4 = (A, C)$

This is a lossless join decomposition.

If R is decomposed into (A, B), (C, D)

This is a lossy-join decomposition.

Dependency Preservation

R - a relation schema

F - set of functional dependencies on R

$\{ R_1, R_2 \}$ – a decomposition of R.

F_i - the set of dependencies in F^+ involves only attributes in R_i.

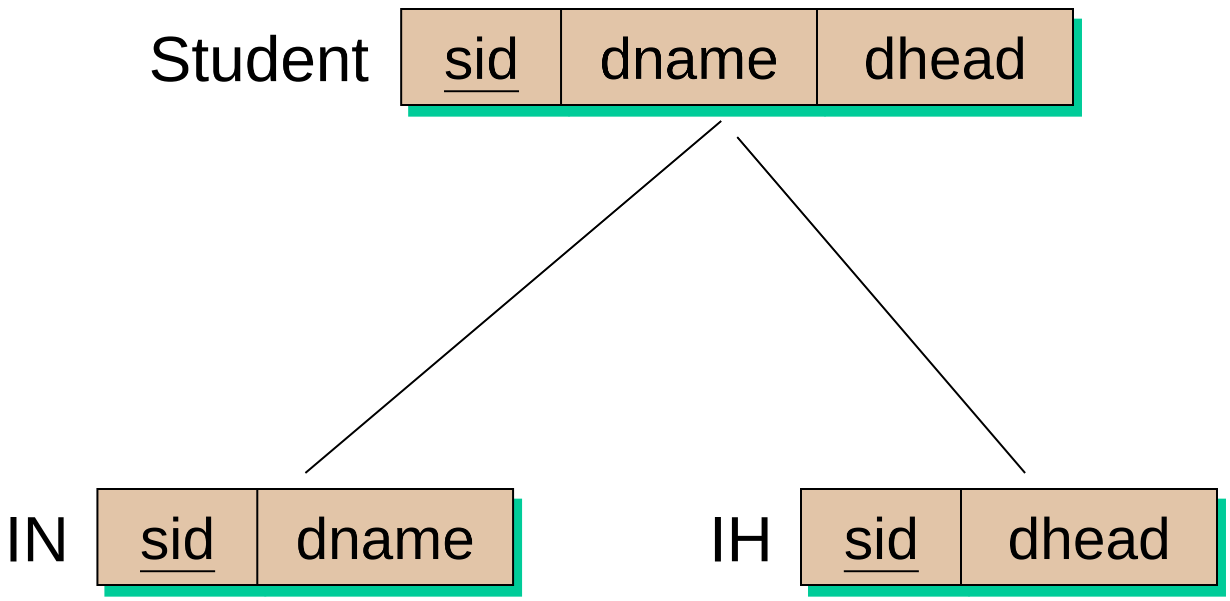
F_i is called the **projection** of F on the set of attributes of R_i.

dependency is preserved if

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

- Intuitively, a dependency-preserving decomposition allows us to enforce all FDs by examining a single relation instance on each insertion or modification of a tuple.

Dependency set: F = { sid $\rightarrow$ dname, dname $\rightarrow$ dhead }



Student

sid	dname	dhead
123	C.S.	Ying Wang
456	C.S.	Ying Wang

IN

sid	dname
123	C.S.
456	C.S.

and

IH

sid	dhead
123	Ying Wang
456	Ying Wang

Consider the RHS update.

Update

This update violates the FD 'dname $\rightarrow$ dhead'. However, it can sometimes only be caught when we join IN and IH

sid	dhead
123	Ying Wang
456	Lin Cheung

IN	<u>sid</u>	dname

IH	<u>sid</u>	dhead

F = { sid $\rightarrow$ dname, dname $\rightarrow$ dhead }

This decomposition does not preserve dependency:

F_{IN} = { trivial dependencies, sid $\rightarrow$ dname,
sid $\rightarrow$ sid dname}

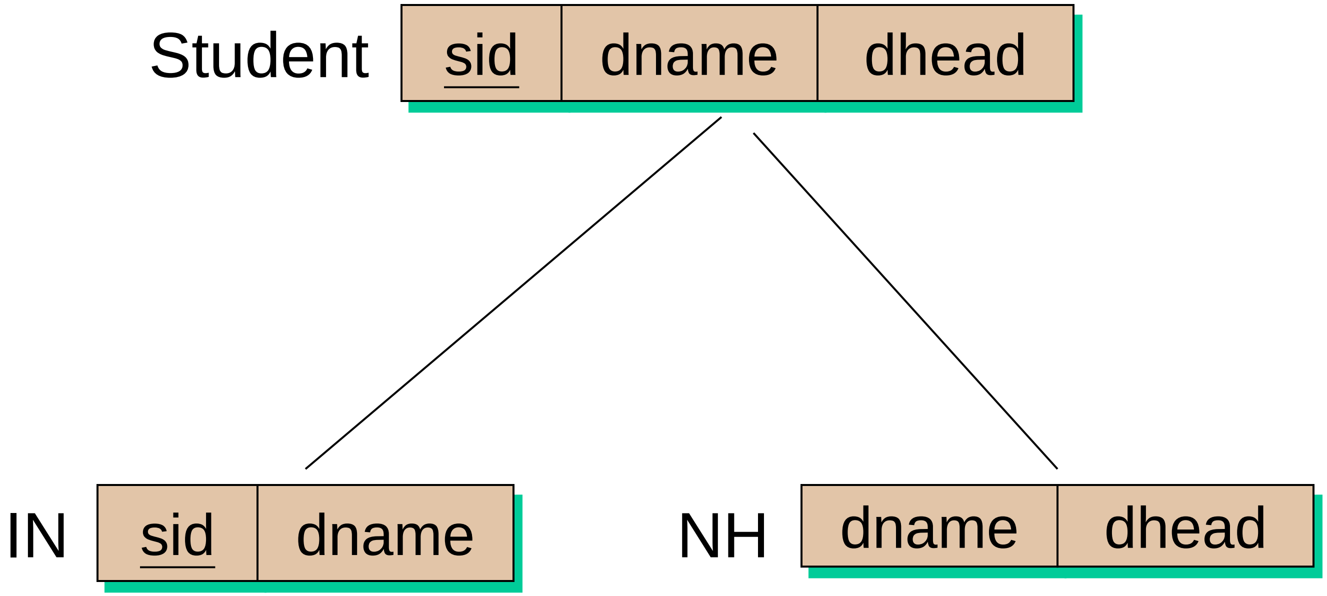
F_{IH} = { trivial dependencies, sid $\rightarrow$ dhead,
sid $\rightarrow$ sid dhead }

We have: dname $\rightarrow$ dhead $\in F^+$ **but**

dname $\rightarrow$ dhead $\notin (F_{IN} \cup F_{IH})^+$

Dependency set: F = { sid $\rightarrow$ dname, dname $\rightarrow$ dhead }

Let's decompose the relation in **another** way.



IN

<u>sid</u>	dname

NH

dname	dhead

F = { sid $\rightarrow$ dname, dname $\rightarrow$ dhead }

This decomposition preserves dependency:

F_{IN} = { trivial dependencies, sid $\rightarrow$ dname,
sid $\rightarrow$ sid dname}

F_{NH} = { trivial dependencies, dname $\rightarrow$ dhead,
dname $\rightarrow$ dname dhead }

$(F_{IN} \cup F_{NH})^{+} = F^{+}$

Student

sid	dname	dhead
123	C.S.	Ying Wang
456	C.S.	Ying Wang

IN

sid	dname
123	C.S.
456	C.S.

and

NH

dname	dhead
C.S.	Ying Wang
C.S.	Ying Wang

Update

dname	dhead
C.S.	Ying Wang
C.S.	Lin Cheung

The error in NH will immediately be caught by the DBMS, since it violates F.D. dname $\rightarrow$ dhead. No join is necessary.