Database and Big Data(2019)

Week 4, S1: Functional dependencies

Cyrille Jegourel

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

DATABASE DESIGN

How do we design a “good” database schema?

We want to ensure the integrity of the data.

We also want to get good performance.

EXAMPLE DATABASE

student(<u>sid</u>, <u>course</u>,room,grade,name,address)

sid	course	room	grade	name	address
123	Philo	LT2	A	Agus	Labrador Park
456	Maths	LT5	B	Bron	Bukit Brown
789	Philo	LT2	A	Hannah	Sungei Buloh
012	Philo	LT2	C	Dewi	Bukit Puaka
789	Maths	LT5	A	Hannah	Sungei Buloh

EXAMPLE DATABASE

student(sid, course,room,grade,name,address)

sid	course	room	grade	name	address
123	Philo	LT2	A	Agus	Labrador Park
456	Maths	LT5	B	Bron	Bukit Brown
789	Philo	LT2	A	Hannah	Sungei Buloh
012	Philo	LT2	C	Dewi	Bukit Puaka
789	Maths	LT5	A	Hannah	Sungei Buloh

REDUNDANCY PROBLEMS

Update Anomalies

→ If the room number changes, we need to make sure that we change all students records.

Insert Anomalies

→ May not be possible to add a student unless they're enrolled in a course.

Delete Anomalies

→ If all the students enrolled in a course are deleted, then we lose the room number.

EXAMPLE DATABASE

student(sid,name,address)

sid	name	address
123	Agus	Labrador Park
456	Bron	Bukit Brown
789	Hannah	Sungei Buloh
012	Dewi	Bukit Puaka

course (sid,course,grade)

Sid	course	grade
123	Philo	A
456	Maths	B
789	Philo	A
012	Maths	C
789	Philo	A

room (course,room)

course	room
Philo	LT2
Maths	LT5

Why is this decomposition better?

How to find it?

- If student graduates, we delete students and course table but we will still have room table.

- Decomposition is more robust.

FUNCTIONAL DEPENDENCIES

A functional dependency is a constraint between two sets of attributes in a relation from a database.

$$X_1, \ldots, X_m \rightarrow Y_1, \ldots, Y_n$$

Antecedent

Consequent

The value of $X_1, \ldots, X_m$ functionally determines the value of $Y_1, \ldots, Y_n$

Given X, we can determine Y.

FUNCTIONAL DEPENDENCIES

R1(<u>sid</u>,name,address)

sid	name	address
123	Agus	Labrador Park
456	Bron	Bukit Brown
789	Hannah	Sungei Buloh
012	Dewi	Bukit Puaka

More formally,
A functional dependency $X_1, \ldots, X_m \rightarrow Y_1, \ldots, Y_n$
holds in a relation R if:

$$\forall t, t' \in R,\ t[X_1] = t'[X_1] \cap \cdots \cap t[X_m] = t'[X_m] \rightarrow t[Y_1] = t'[Y_1] \cap \cdots \cap t[Y_n] = t'[Y_n]$$

FUNCTIONAL DEPENDENCIES

e.g.,
$\mathbf{X_1}$: sid
$\mathbf{Y_1}$,$\mathbf{Y_2}$: name, address

R1(sid,name,address)

sid	name	address
123	Agus	Labrador Park
456	Bron	Bukit Brown
789	Hannah	Sungei Buloh
012	Dewi	Bukit Puaka
123	Agus	Labrador Park

More formally,
A functional dependency $\mathbf{X_1}, \ldots, X_m \rightarrow \mathbf{Y_1}, \ldots, Y_n$
holds in a relation R if:

$$\forall t, t' \in R,\ \boldsymbol{t}[\mathbf{X_1}] = \boldsymbol{t}'[\mathbf{X_1}] \cap \cdots \cap t[X_m] = t'[X_m] \rightarrow \boldsymbol{t}[\mathbf{Y_1}] = \boldsymbol{t}'[\mathbf{Y_1}] \cap \cdots \cap t[Y_n] = t'[Y_n]$$

FUNCTIONAL DEPENDENCIES

R1(sid,name,address)

sid	name	address
123	Agus	Labrador Park
456	Bron	Bukit Brown
789	Hannah	Sungei Buloh
012	Dewi	Bukit Puaka

More formally,
A functional dependency $X_1, \ldots, X_m \rightarrow Y_1, \ldots, Y_n$
holds in a relation *R* if:

$$\forall t, t' \in R,\ t[X_1] = t'[X_1] \cap \cdots \cap t[X_m] = t'[X_m] \rightarrow t[Y_1] = t'[Y_1] \cap \cdots \cap t[Y_n] = t'[Y_n]$$

✓ sid → name

FUNCTIONAL DEPENDENCIES

FD is a constraint that allows instances for which the FD holds

You can check (1) if a FD is violated by an instance, (2) but you cannot prove that a FD is part of the schema using an instance.

e.g.:

(1) if **sid** → **name** is given to us, we can identify a violation on the right table.

(2) From the initial table we could not deduce a FD: **name** → **address**

R1(sid,name,address)

sid	name	address
123	Agus	Labrador Park
456	Bron	Bukit Brown
789	Hannah	Sungei Buloh
012	Dewi	Bukit Puaka
555	Agus	East Coast Park
456	Briac	Bukit Brown

FUNCTIONAL DEPENDENCIES

Two FDs $X \rightarrow Y$ and $X \rightarrow Z$ can be written in shorthand as $X \rightarrow YZ$.

But $XY \rightarrow Z$ is not the same as the two FDs $X \rightarrow Z$ and $Y \rightarrow Z$.

WHY SHOULD I CARE?

FDs seem important, but what can we actually do with them?

They allow us to decide whether a database design is correct.

→ Note that this different then the question of whether it's a good idea for performance…

IMPLIED DEPENDENCIES

student(sid, course,room,grade,name,address)

sid	course	room	grade	name	address
123	Philo	LT2	A	Agus	Labrador Park
456	Maths	LT5	B	Bron	Bukit Brown
789	Philo	LT2	A	Hannah	Sungei Buloh
012	Philo	LT2	C	Dewi	Bukit Puaka

Provided FDs

sid → name, address
sid,course → grade

Implied FDs

name,address,course → grade
sid,course → sid
sid,course → course

IMPLIED DEPENDENCIES

Given a set of FDs $\{f_1, \ldots, f_n\}$, how do we decide whether a FD g holds?

Compute the closure using Armstrong's axioms which is the set of all implied FDs.

ARMSTRONG'S AXIOMS (Primary rules)

Reflexivity:

$$Y \subseteq X \Rightarrow X \longrightarrow Y$$

e.g., {sid} ⊆ {sid,course}, so, {sid,course}→ {sid}

Augmentation:

$$X \longrightarrow Y \Rightarrow XZ \longrightarrow YZ$$

e.g., sid→ name, so, sid,course→ name,course

Transitivity:

$$X \longrightarrow Y \wedge Y \longrightarrow Z \Rightarrow X \longrightarrow Z$$

e.g., sid,course→ sid and sid→ name, so sid,course→ name

ARMSTRONG'S AXIOMS (Secundary rules)

Union:

$$(X \longrightarrow Y) \wedge (X \longrightarrow Z) \Longrightarrow X \longrightarrow YZ$$

e.g., **sid**→ **name** **and** **sid**→ **address**, **so** **sid**→ **name,address**

Decomposition

$$X \longrightarrow YZ \Longrightarrow (X \longrightarrow Y) \wedge (X \longrightarrow Z)$$

e.g., **sid**→ **name,address**, **so**, **sid**→ **name** **and** **sid**→ **address**

Pseudo-transitivity

$X \longrightarrow Y \wedge YW \longrightarrow Z \Longrightarrow XW \longrightarrow Z$

e.g., **sid**→ **sid** **and** **sid,course**→ **grade**, **so** **sid,course**→ **grade**

Why?

sid -> sid
reflexivity rule used.

CLOSURES

Given a set **F** of FDs $\{f_1, \dots, f_n\}$, we define the closure **F**$^+$ is the set of all implied FDs.

student(<u>sid</u>, <u>course</u>,room,grade,name,address)

F
- sid,course → grade
- sid → name, address

Reflexivity →	sid,name → sid
Decomposition →	sid → name
Decomposition →	sid → address
Augmentation →	sid,grade → name, grade
Transitivity →	sid,course → grade,name
	⋮

F$^+$

WHY DO WE NEED THE CLOSURE?

With the closure we can find all FD's easily and then compute the attribute closure:

→ For a given attribute X, the closure X^+ is the set of all attributes such that $X \rightarrow A$ can be inferred using the Armstrong's Axioms.

To check if $X \rightarrow A$:

→ Compute X^+

→ Check if $A \in X^+$

BUT AGAIN, WHY SHOULD I CARE?

Maintaining the closure at runtime is expensive:

→ The DBMS has to check all the constraints for every **INSERT**, **UPDATE**, and **DELETE** operation.

We want a minimal set of FDs that was enough to ensure correctness.

CANONICAL COVER

Given a set **F** of FDs $\{f_1, \ldots, f_n\}$, we define the <u>canonical cover</u> F_c as the minimal set of all FDs.

F

F_c

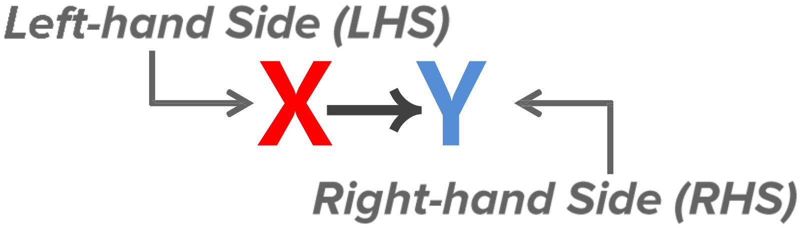
sid,course → grade
sid → name, address

sid,name → name, address
sid,course → grade,name

CANONICAL COVER DEFINITION

A canonical cover F_c must have the following properties:

1. The RHS of every FD is a single attribute.
2. The closure of F_c is identical to the closure of F (i.e., $F_c^+ = F^+$ are equivalent).
3. The F_c is minimal (i.e., if we eliminate any attribute from the LHS or RHS of a FD, property #2 is violated.



COMPUTING THE CANONICAL COVER

Given a set **F** of FDs, examine each FD:

→ Drop extraneous LHS or RHS attributes; or redundant FDs.

→ Make sure that FDs have a single attribute in their RHS.

Repeat until no change.

Left-hand Side (LHS)

$X \to Y$

Right-hand Side (RHS)

COMPUTING THE CANONICAL COVER (1)

F:

$AB \rightarrow C \quad (1)$

$A \rightarrow BC \quad (2)$

$B \rightarrow C \quad (3)$

$A \rightarrow B \quad (4)$

COMPUTING THE CANONICAL COVER (1)

F:

$AB \rightarrow C$	(1)
$\boxed{A \rightarrow BC}$	(2)
$B \rightarrow C$	(3)
$A \rightarrow B$	(4)

Split (2) →

F_1:

$AB \rightarrow C$	(1)
$\boxed{A \rightarrow B}$	(2′)
$\boxed{A \rightarrow C}$	(2′′)
$B \rightarrow C$	(3)
$A \rightarrow B$	(4)

COMPUTING THE CANONICAL COVER (2)

F_1:

$AB \rightarrow C$	(1)
$\boxed{A \rightarrow B}$	(2′)
$A \rightarrow C$	(2′′)
$B \rightarrow C$	(3)
$\boxed{A \rightarrow B}$	(4)

Eliminate (2′) →

F_2:

$AB \rightarrow C$	(1)
$A \rightarrow C$	(2′′)
$B \rightarrow C$	(3)
$A \rightarrow B$	(4)

COMPUTING THE CANONICAL COVER (3)

F_2:	
$AB \rightarrow C$	(1)
$\boxed{A \rightarrow C}$	(2′′)
$B \rightarrow C$	(3)
$A \rightarrow B$	(4)

Eliminate (2″) →

F_3:	
$AB \rightarrow C$	(1)
$B \rightarrow C$	(3)
$A \rightarrow B$	(4)

COMPUTING THE CANONICAL COVER (4)

F_3:

$\boxed{A}B \rightarrow C$ (1)

$B \rightarrow C$ (3)

$A \rightarrow B$ (4)

Eliminate A from (1)

F_4:

$B \rightarrow C$ (1')

$B \rightarrow C$ (3)

$A \rightarrow B$ (4)

COMPUTING THE CANONICAL COVER (5)

F_4:

$\boxed{B \rightarrow C}$ (1')

$\boxed{B \rightarrow C}$ (3)

$A \rightarrow B$ (4)

Eliminate (1')

F_5:

$B \rightarrow C$ (3)

$A \rightarrow B$ (4)

COMPUTING THE CANONICAL COVER (5)

F_c

F_4:

$\boxed{B \rightarrow C}$ (1')

$\boxed{B \rightarrow C}$ (3)

$A \rightarrow B$ (4)

Eliminate (1')

F_5:

$B \rightarrow C$ (3)

$A \rightarrow B$ (4)

- ✓ Nothing is extraneous
- ✓ All RHS are single attributes
- ✓ Final & original set of FDs are equivalent (same closure)

NO REALLY, WHY SHOULD I CARE?

The canonical cover is the minimum number of assertions that we need to implement to make sure that our database integrity is correct.

It allows us to find the super key for a relation.

RELATIONAL MODEL: KEYS (1)

Super Key:

→ Any set of attributes in a relation that functionally determines all attributes in the relation.

Candidate Key:

→ Any super key such that the removal of any attribute leaves a set that does not functionally determine all attributes.

RELATIONAL MODEL: KEYS (2)

Super Key:

→ Set of attributes for which there are no two distinct tuples with the same values for the attributes in this set.

Candidate Key:

→ Set of attributes that uniquely identifies a tuple according to a key constraint.

RELATIONAL MODEL: KEYS (3)

Super Key:

→ A set of attributes that uniquely identifies a tuple.

Candidate Key:

→ A minimal set of attributes that uniquely identifies a tuple.

Primary Key:

→ Usually just the candidate key.

RELATIONAL MODEL: KEYS

student(<u>sid</u>, <u>course</u>,room,grade,name,address)

sid	course	room	grade	name	address
123	Philo	LT2	A	Agus	Labrador Park
456	Maths	LT5	B	Bron	Bukit Brown
789	Philo	LT2	A	Hannah	Sungei Buloh
012	Philo	LT2	C	Dewi	Bukit Puaka

Provided FDs

sid → name, address
sid,course → grade
course → room

sid,course,room is a **superkey**.
sid,course is a **candidate key**, that we can choose as our **primary key**. When a key has several attributes, we say that it is a **composite key**.

BUT WHY CARE ABOUT SUPER KEYS?

They help us determine whether it is okay to decompose a table into multiple sub-tables.

Super keys ensure that we are able to recreate the original relation through joins.

SCHEMA DECOMPOSITIONS

Split a single relation **R** into a set of relations **$\{R_1,\ldots,R_n\}$**.

Not all decompositions make the database schema better:

→ Update Anomalies
→ Insert Anomalies
→ Delete Anomalies
→ Wasted Space

DECOMPOSITION GOALS

Loseless Joins

→ Want to be able to reconstruct original relation by joining smaller ones using a natural join.

Dependency Preservation

→ Want to minimize the cost of global integrity constraints based on FD's.

Redundancy Avoidance

→ Avoid unnecessary data duplication.

DECOMPOSITION GOALS

Loseless Joins

→ Want to be able to reconstruct original relation by joining smaller ones using a natural join.

← **Mandatory!**

Dependency Preservation

→ Want to minimize the cost of global integrity constraints based on FD's.

Redundancy Avoidance

→ Avoid unnecessary data duplication.

← **Nice to have, but not required**

LOSSLESS DECOMPOSITION (1)

Provided FDs

bname → bcity, assets

loanId → amt,bname

loans(bname,bcity,assets,cname,loanId,amt)

bname	bcity	assets	cname	loanId	amt
Downtown	Pittsburgh	$9M	Andy	L-17	$1000
Downtown	Pittsburgh	$9M	Oswin	L-23	$2000
Compton	Los Angeles	$2M	Andy	L-93	$500
Downtown	Pittsburgh	$9M	Damian	L-17	$1000

LOSSLESS DECOMPOSITION (1)

Provided FDs
bname → bcity, assets
loanId → amt,bname

loans (bname,bcity,assets,cname,loanId,amt)

R1(bname,bcity,assets,cname)

bname	bcity	assets	cname
Downtown	Pittsburgh	$9M	Andy
Downtown	Pittsburgh	$9M	Oswin
Compton	Los Angeles	$2M	Andy
Downtown	Pittsburgh	$9M	Damian

R2(cname,loanId,amt)

cname	loanId	amt
Andy	L-17	$1000
Oswin	L-23	$2000
Andy	L-93	$500
Damian	L-17	$1000

LOSSLESS DECOMPOSITION (1)

Provided FDs
bname → bcity, assets
loanId → amt,bname

R1(bname,bcity,assets,cname)

bname	bcity	assets	cname
Downtown	Pittsburgh	$9M	Andy
Downtown	Pittsburgh	$9M	Oswin
Compton	Los Angeles	$2M	Andy
Downtown	Pittsburgh	$9M	Damian

⋈

R2(cname,loanId,amt)

cname	loanId	amt
Andy	L-17	$1000
Oswin	L-23	$2000
Andy	L-93	$500
Damian	L-17	$1000

LOSSLESS DECOMPOSITION (1)

Provided FDs

bname → bcity, assets

loanId → amt,bname

R1(bname,bcity,assets,cname)

bname	bcity	assets	cname
Downtown	Pittsburgh	$9M	Andy
Downtown	Pittsburgh	$9M	Oswin
Compton	Los Angeles	$2M	Andy
Downtown	Pittsburgh	$9M	Damian

$\bowtie$

R2(cname,loanId,amt)

cname	loanId	amt
Andy	L-17	$1000
Oswin	L-23	$2000
Andy	L-93	$500
Damian	L-17	$1000

bname	bcity	assets	cname	loanId	amt
Downtown	Pittsburgh	$9M	Andy	L-17	$1000
Downtown	Pittsburgh	$9M	Andy	L-93	$500
Downtown	Pittsburgh	$9M	Oswin	L-23	$2000
Compton	Los Angeles	$2M	Andy	L-17	$1000
Compton	Los Angeles	$2M	Andy	L-93	$500
Downtown	Pittsburgh	$9M	Damian	L-17	$1000

LOSSLESS DECOMPOSITION (2)

Provided FDs

bname → bcity, assets

loanId → amt,bname

R1(bname,bcity,assets,cname)

bname	bcity	assets	cname
Downtown	Pittsburgh	$9M	Andy
Downtown	Pittsburgh	$9M	Oswin
Compton	Los Angeles	$2M	Andy
Downtown	Pittsburgh	$9M	Damian

$\bowtie$

R2(bname,loanId,amt)

bname	loanId	amt
Downtown	L-17	$1000
Downtown	L-23	$2000
Compton	L-93	$500

LOSSLESS DECOMPOSITION (2)

R1(bname,bcity,assets,cname)

bname	bcity	assets	cname
Downtown	Pittsburgh	$9M	Andy
Downtown	Pittsburgh	$9M	Obama
Compton	Los Angeles	$2M	Andy
Downtown	Pittsburgh	$9M	Damian

R2(cname,loanId,amt)

bname	loanId	amt
Downtown	L-17	$1000
Downtown	L-23	$2000
Compton	L-93	$500

$\bowtie$

bname	bcity	assets	cname	loanId	amt
Downtown	Pittsburgh	$9M	Andy	L-17	$1000
Downtown	Pittsburgh	$9M	Andy	L-23	$2000
Downtown	Pittsburgh	$9M	Oswin	L-17	$1000
Downtown	Pittsburgh	$9M	Oswin	L-23	$2000
Compton	Los Angeles	$2M	Andy	L-93	$500
Downtown	Pittsburgh	$9M	Damian	L-23	$1000

LOSSLESS DECOMPOSITION (3)

Provided FDs

bname → bcity, assets

loanId → amt,bname

R1(bname,assets,cname,loanId)

bname	assets	cname	loanId
Downtown	$9M	Andy	L-17
Downtown	$9M	Oswin	L-23
Compton	$2M	Andy	L-93
Downtown	$9M	Damian	L-17

⋈

R2(loanId,bcity,amt)

loanId	bcity	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Los Angeles	$500

LOSSLESS DECOMPOSITION (3)

Provided FDs

bname → bcity, assets

loanId → amt,bname

R1(bname,assets,cname,loanId)

bname	assets	cname	loanId
Downtown	$9M	Andy	L-17
Downtown	$9M	Oswin	L-23
Compton	$2M	Andy	L-93
Downtown	$9M	Damian	L-17

⋈

R2(loanId,bcity,amt)

loanId	bcity	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Los Angeles	$500

✓

bname	bcity	assets	cname	loanId	amt
Downtown	Pittsburgh	$9M	Andy	L-17	$1000
Downtown	Pittsburgh	$9M	Oswin	L-23	$2000
Compton	Los Angeles	$2M	Andy	L-93	$500
Downtown	Pittsburgh	$9M	Damian	L-17	$1000

DEPENDENCY PRESERVATION

A schema preserves dependencies if its original FDs do not span multiple tables.

Why does this matter?

→ It would be expensive to check (assuming that our DBMS supports ASSERTIONS).

DEPENDENCY PRESERVATION (1)

R1(bname,assets,cname,loanId)

bname	assets	cname	loanId
Downtown	$9M	Andy	L-17
Downtown	$9M	Oswin	L-23
Compton	$2M	Andy	L-93
Downtown	$9M	Damian	L-17

R2(loanId,bcity,amt)

loanId	bcity	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Los Angeles	$500

Provided FDs

bname → bcity,assets

loanId → amt,bname

DEPENDENCY PRESERVATION (1)

R1(bname,assets,cname,loanId)

bname	assets	cname	loanId
Downtown	$9M	Andy	L-17
Downtown	$9M	Oswin	L-23
Compton	$2M	Andy	L-93
Downtown	$9M	Damian	L-17

R2(loanId,bcity,amt)

loanId	bcity	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Los Angeles	$500

canonical form
loadid -> bname
bname -> bcity
bname -> assets
loadid -> amt

Provided FDs
bname → bcity,assets
loanId → amt,bname

DEPENDENCY PRESERVATION (1)

R1(bname,assets,cname,loanId)

bname	assets	cname	loanId
Downtown	$9M	Andy	L-17
Downtown	$9M	Obama	L-23
Compton	$2M	Andy	L-93
Downtown	$9M	DJ Snake	L-17

R2(loanId,bcity,amt)

loanId	bcity	amt
L-17	Pitt burgh	$1000
L-23	Pitt burgh	$2000
L-93	Los Angeles	$500

Provided FDs

bname → bcity,assets

loanId → amt,bname

DEPENDENCY PRESERVATION

To test whether the decomposition $R=\{R_1,\ldots,R_n\}$ preserves the FD set F:

→ Compute F^+

→ Compute G as the union of the set of FDs in F^+ that are covered by $\{R_1,\ldots,R_n\}$

→ Compute G^+

→ If $F^+ = G^+$, then $\{R_1,\ldots,R_n\}$ is Dependency Preserving

DEPENDENCY PRESERVATION (2)

Is $R=\{R_1,R_2\}$ dependency preserving?

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

R1(A,B,C) R2(C,D)

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

DEPENDENCY PRESERVATION (2)

Is $R=\{R_1,R_2\}$ dependency preserving?

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

$G = \underbrace{\boxed{\{A \rightarrow B\}}}_{\text{FDs covered by } R_1} \cup \underbrace{\boxed{\{C \rightarrow D\}}}_{\text{FDs covered by } R_2}$

R1(A,B,C) R2(C,D)

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

DEPENDENCY PRESERVATION (2)

Is $R=\{R_1,R_2\}$ dependency preserving?

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

$G = \{A \rightarrow B\} \cup \{C \rightarrow D\}$

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

R1(A,B,C) R2(C,D)

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

DEPENDENCY PRESERVATION (2)

Is $R=\{R_1,R_2\}$ dependency preserving?

$(A \to D) \in F^+$

$F^+ = \{A \to B, AB \to D, \boxed{A \to D}, C \to D\}$

$G = \{A \to B\} \cup \{C \to D\}$

$G^+ = \{A \to B, C \to D\}$

$F^+ \neq \boxed{G^+}$ because $(A \to D) \in (F^+ - G^+)$

$(A \to D) \notin G^+$

$R1(A,B,C) \quad R2(C,D)$

$F = \{A \to B, AB \to D, C \to D\}$

Decomposition is <u>not</u> DP

DEPENDENCY PRESERVATION (3)

Is $R=\{R_1,R_2\}$ dependency preserving?

R1(A,B,D) R2(C,D)

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

DEPENDENCY PRESERVATION (3)

Is $R=\{R_1,R_2\}$ dependency preserving?

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

$G = \{A \rightarrow B, A \rightarrow D, AB \rightarrow D\} \cup \{C \rightarrow D\}$

$G^+ = \{A \rightarrow B, AB \rightarrow D, A \rightarrow D, C \rightarrow D\}$

$F^+ = G^+$

R1(A,B,D) R2(C,D)

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

Decomposition is DP

DECOMPOSITION SUMMARY

Lossless Joins

- Motivation: Avoid information loss
- Goal: No noise introduced when reconstituting the main relation via joins
- Test: we will see that on Wednesday (chasing algorithm).

DECOMPOSITION SUMMARY

Dependency Preservation

- Motivation: asserting efficient FD
- Goal: guaranteeing that all the FDs specified in F are preserved at minimal cost in the decomposition.
- Test: $\mathrm{R} = (R_1 \cup \cdots \cup Rn)$ is dependency preserving if the closure of FDs covered by each R_i = closure of the initial FDs covered in R.

CONCLUSION

Functional dependencies are simple to understand.

They will allow us to reason about schema decompositions.