Backend Development– Complement: Functional Dependencies and Normal Forms

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(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

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?

TODAY'S AGENDA

Functional Dependencies
Canonical Cover
Schema Decomposition

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 $Y_1, \ldots, Y_n$

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 $\mathbf{X_1}, \ldots, X_m \rightarrow \mathbf{Y_1}, \ldots, Y_n$ 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(<u>sid</u>,name,address)

e.g.,
X_1 : sid
Y_1,Y_2 : 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 $X_1, \dots, X_m \rightarrow Y_1, \dots, Y_n$ 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

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 $X_1, \dots, X_m \rightarrow Y_1, \dots, Y_n$ 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:

$$Y \subseteq X \implies X \longrightarrow Y$$

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

Augmentation:

$$X \longrightarrow Y \implies XZ \longrightarrow YZ$$

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

Transitivity:

$$X \longrightarrow Y \wedge Y \longrightarrow Z \implies X \longrightarrow Z$$

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

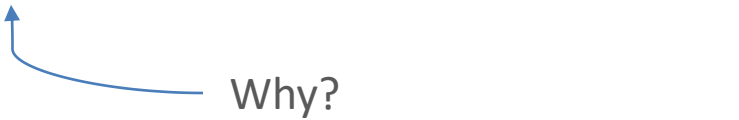
$$X \longrightarrow YZ \Longrightarrow (X \longrightarrow Y) \wedge (X \longrightarrow Z)$$

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**



CLOSURES

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

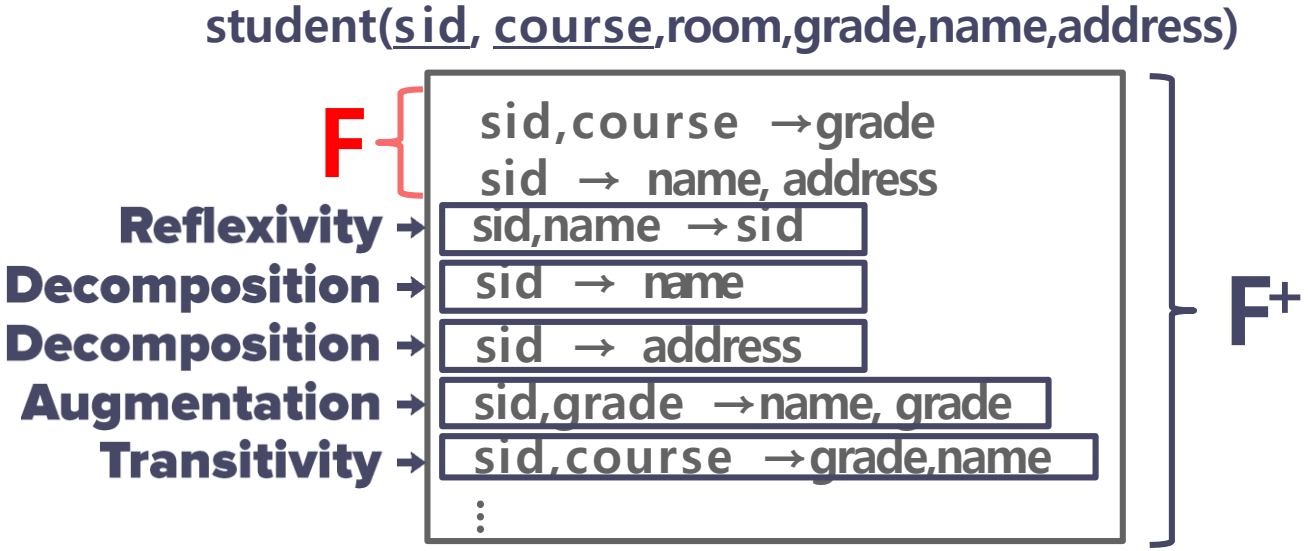
student(<u>sid</u>, <u>course</u>,room,grade,name,address)

F

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

CLOSURES

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



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?

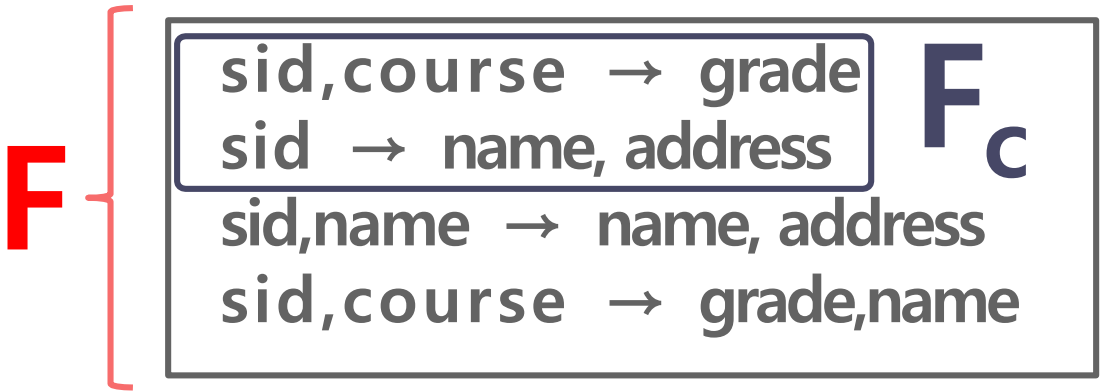
Maintaining the closure at runtime is expensive:

→ 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

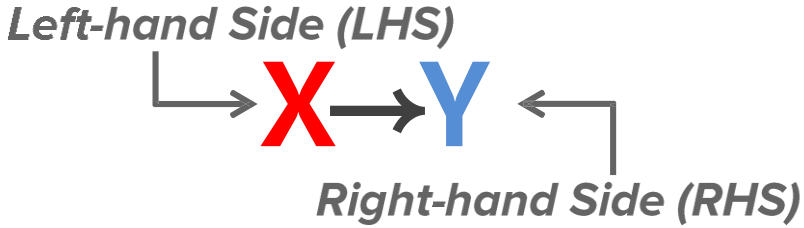
Given a set **F** of FDs $\{f_1, \ldots, f_n\}$, we define the canonical cover $\mathbf{F_c}$ as the minimal set of all FDs.



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.



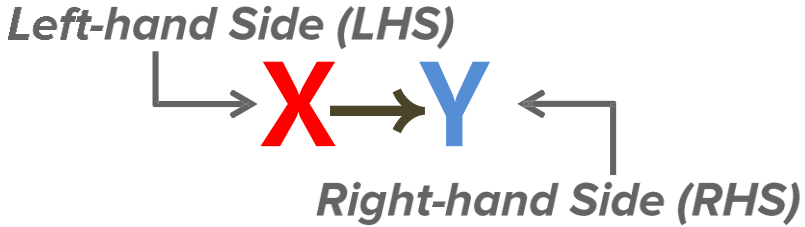
COMPUTING THE CANONICAL COVER

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.



COMPUTING THE CANONICAL COVER (1)

F:

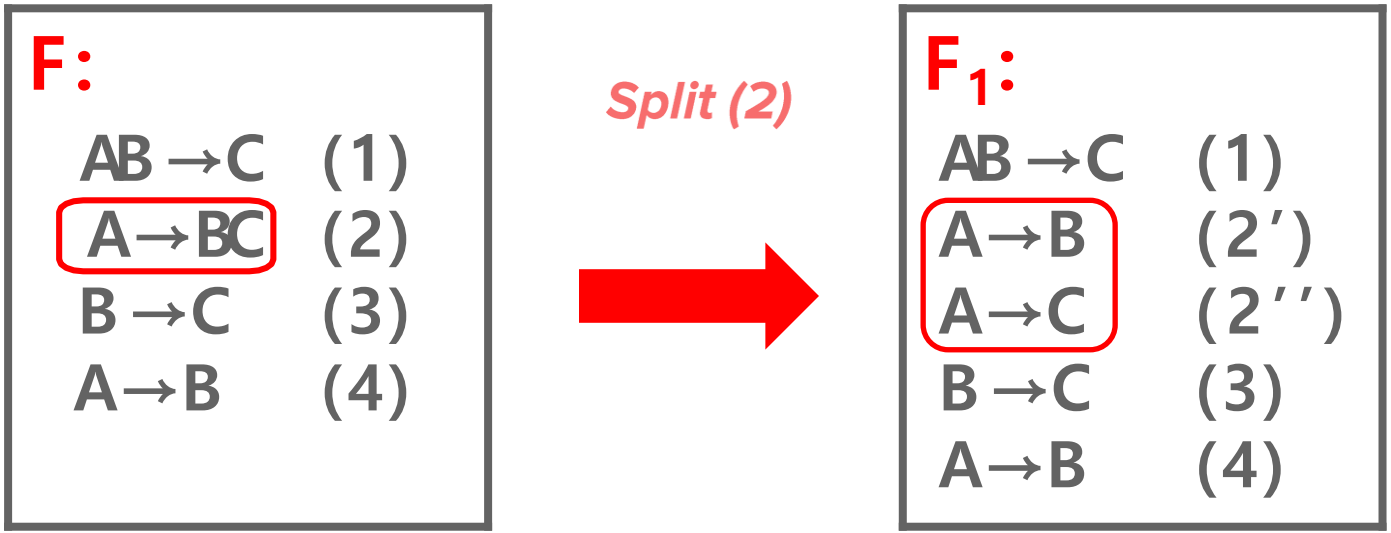
$AB \rightarrow C$ (1)

$A \rightarrow BC$ (2)

$B \rightarrow C$ (3)

$A \rightarrow B$ (4)

COMPUTING THE CANONICAL COVER (1)



COMPUTING THE CANONICAL COVER (2)

F_1:

AB → C	(1)
A → B	(2′)
A → C	(2′′)
B → C	(3)
A → B	(4)

Eliminate (2')

F_2:

AB → C	(1)
A → C	(2′′)
B → C	(3)
A → B	(4)

COMPUTING THE CANONICAL COVER (3)

F_2:

AB → C	(1)
A → C	(2'')
B → C	(3)
A → B	(4)

Eliminate (2'')

F_3:

AB → C	(1)
B → C	(3)
A → B	(4)

COMPUTING THE CANONICAL COVER (4)

F_3:

AB → C	(1)
B → C	(3)
A → B	(4)

Eliminate A from (1)

F_4:

B →	C	(1')
B →	C	(3)
A →	B	(4)

COMPUTING THE CANONICAL COVER (5)

F_4:

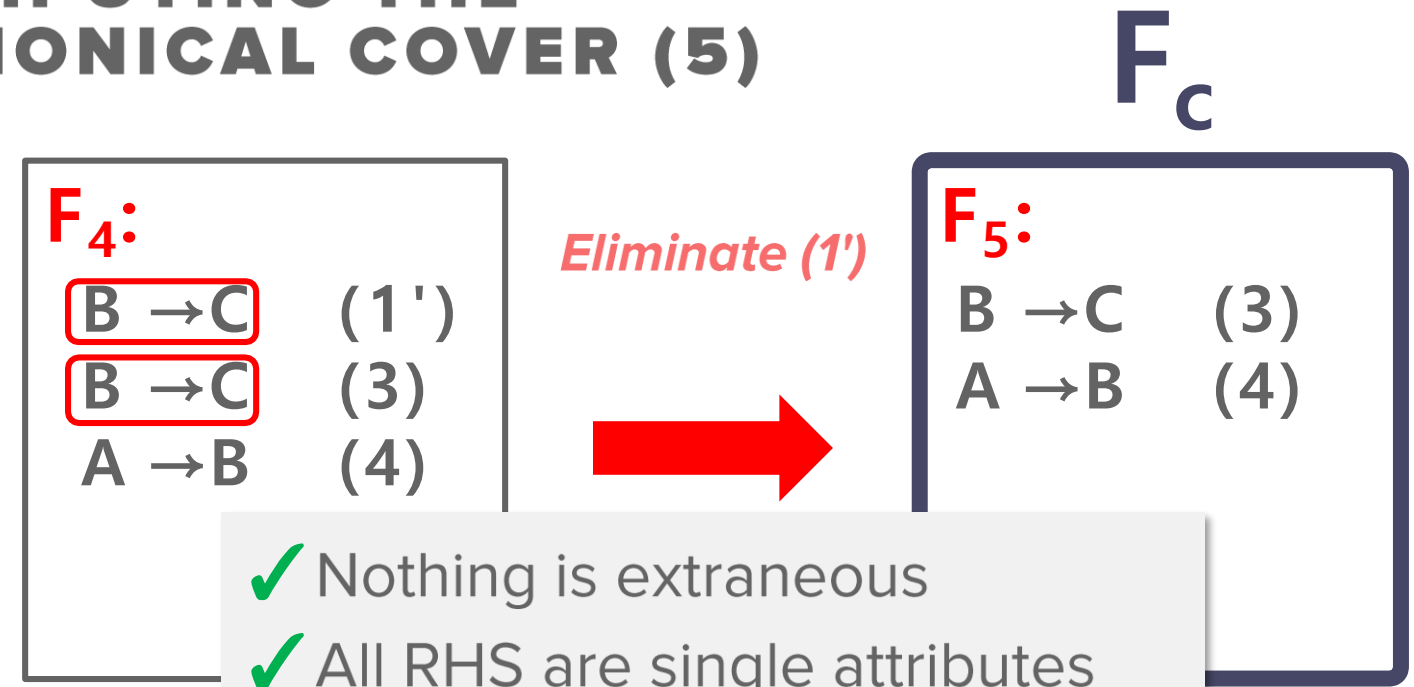
B → C	(1')
B → C	(3)
A → B	(4)

Eliminate (1')

F_5:

B → C	(3)
A → B	(4)

COMPUTING THE CANONICAL COVER (5)



- ✓ 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(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

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)

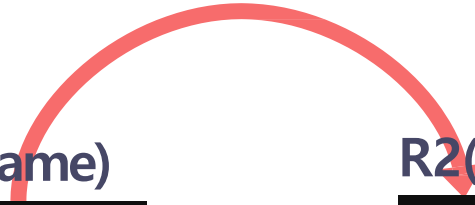
bname	bcity	assets	cname
Downtown	Pittsburgh	$9M	Andy
Downtown	Pittsburgh	$9M	Oswin
Compton	Los Angeles	$2M	Andy
Downtown	Pittsburgh	$9M	Damian

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

⋈

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

⋈

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

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)

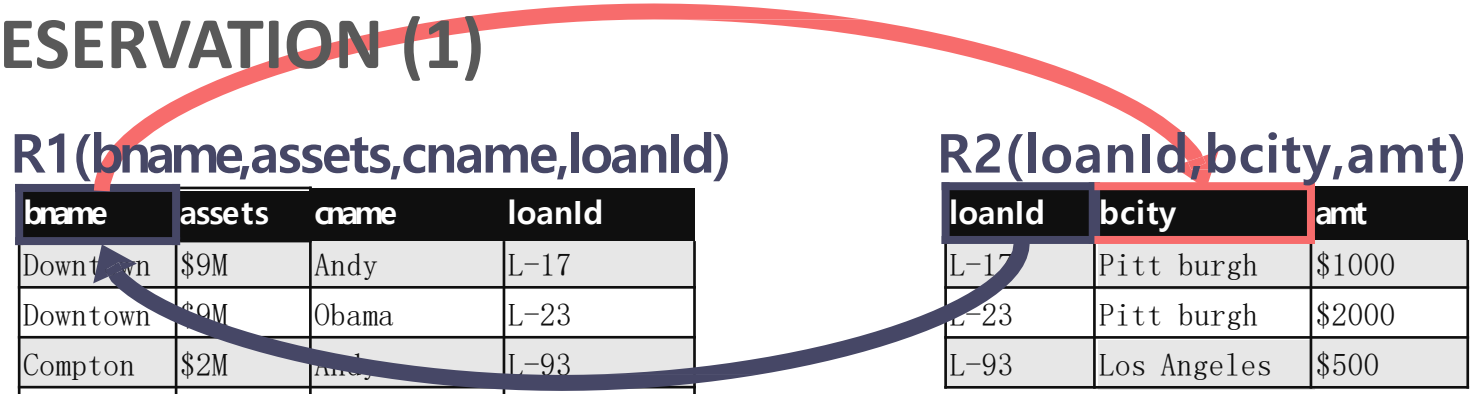
loanId	bcity	amt
L-17	Pittsburgh	$1000
L-23	Pittsburgh	$2000
L-93	Los Angeles	$500

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	[illegible]	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?

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

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

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

DEPENDENCY PRESERVATION (2)

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

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

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

FDs covered by R_1 | FDs covered by R_2

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

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

DEPENDENCY PRESERVATION (2)

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

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

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

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

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

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

DEPENDENCY PRESERVATION (2)

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

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

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

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

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

$F^+ \neq G^+$ because $(A \rightarrow D) \in (F^+ - G^+)$

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

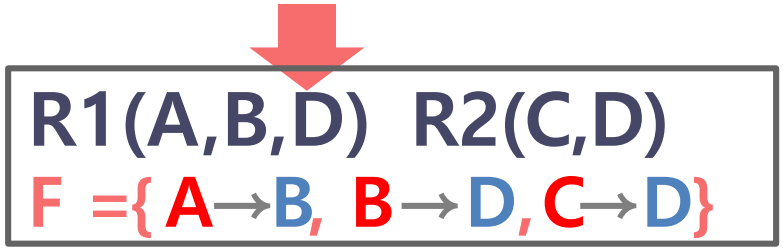
R1(A,B,C) R2(C,D)

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

Decomposition is not DP

DEPENDENCY PRESERVATION (3)

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



DEPENDENCY PRESERVATION (3)

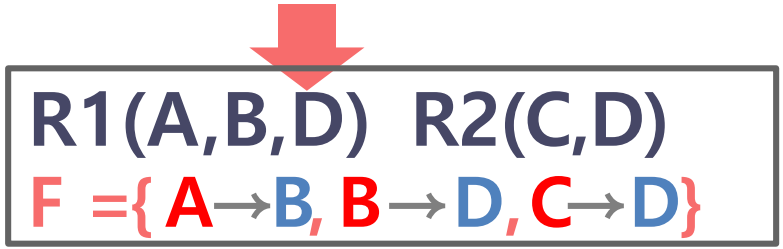
Is $R=\{R_1,R_2\}$ dependency preserving?

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

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

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

$F^+ = G^+$



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.

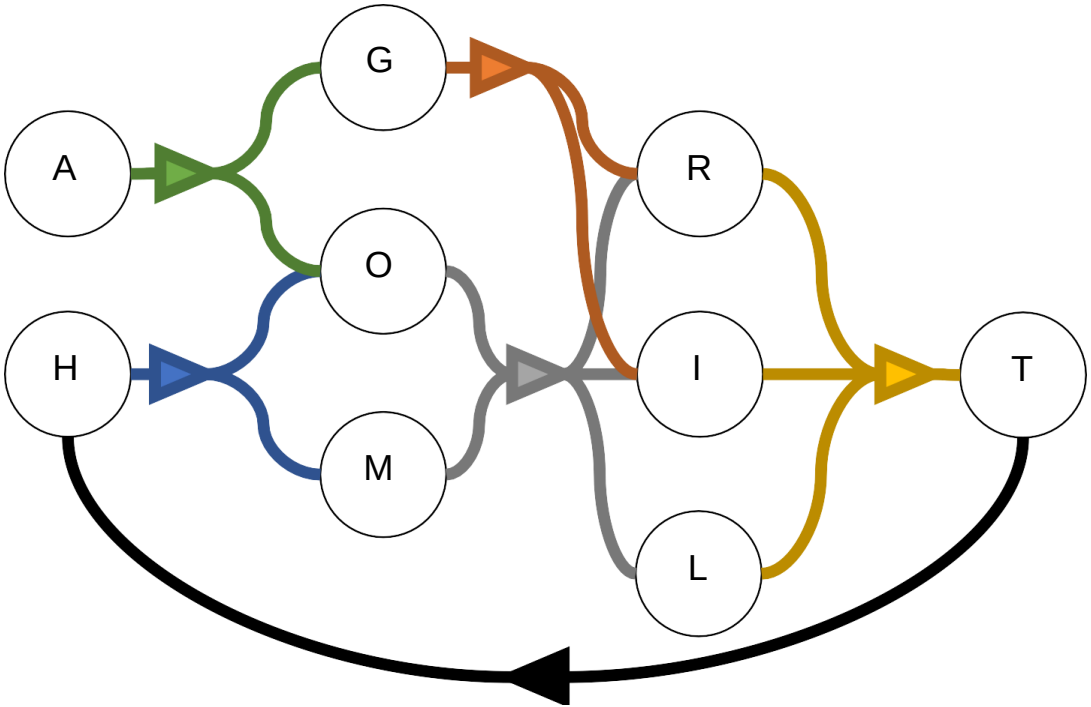
They will allow us to reason about schema decompositions.

NORMAL FORMS

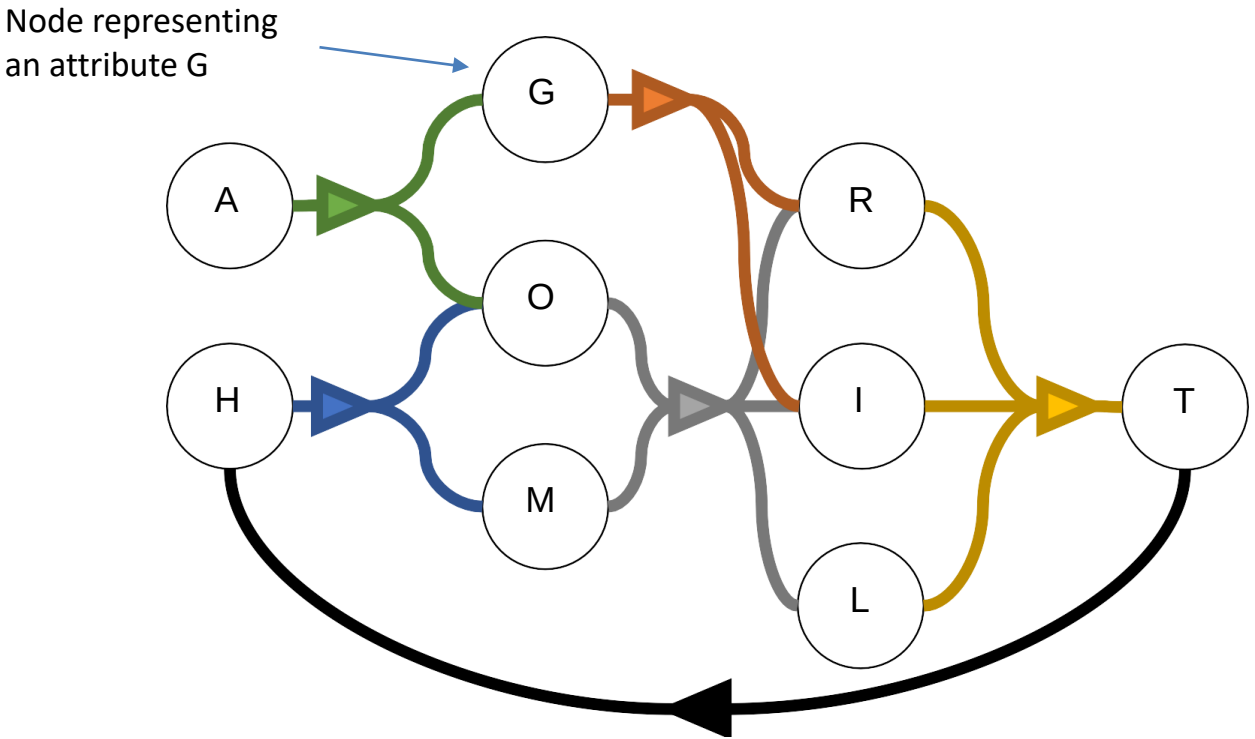
OUTLINE

Visualization of FDs
Normalization: 1NF, 2NF, 3NF, BCNF, 4NF…
Losslessness
Chase method to verify the losslessness of a decomposition.

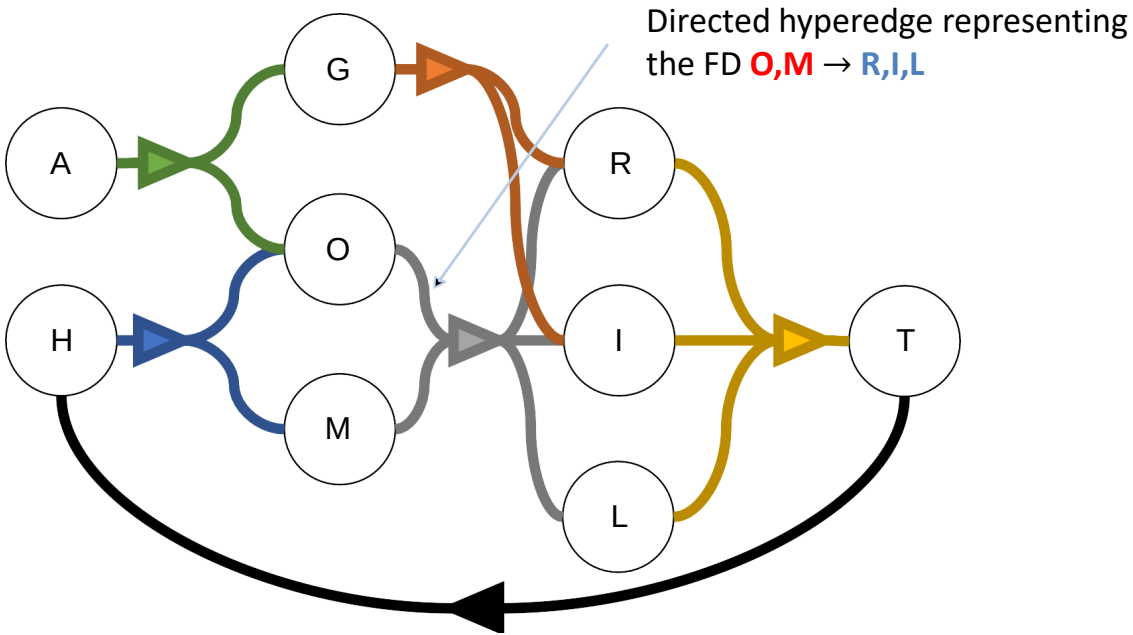
Visualization of FDs: directed hypergraphs



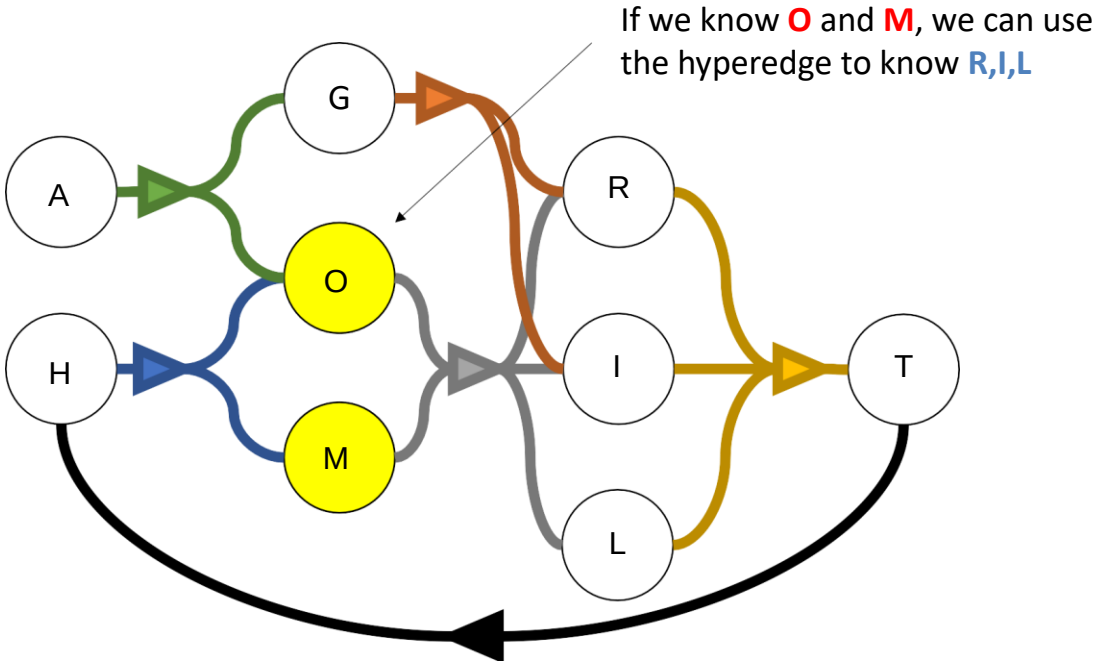
Visualization of FDs: directed hypergraphs



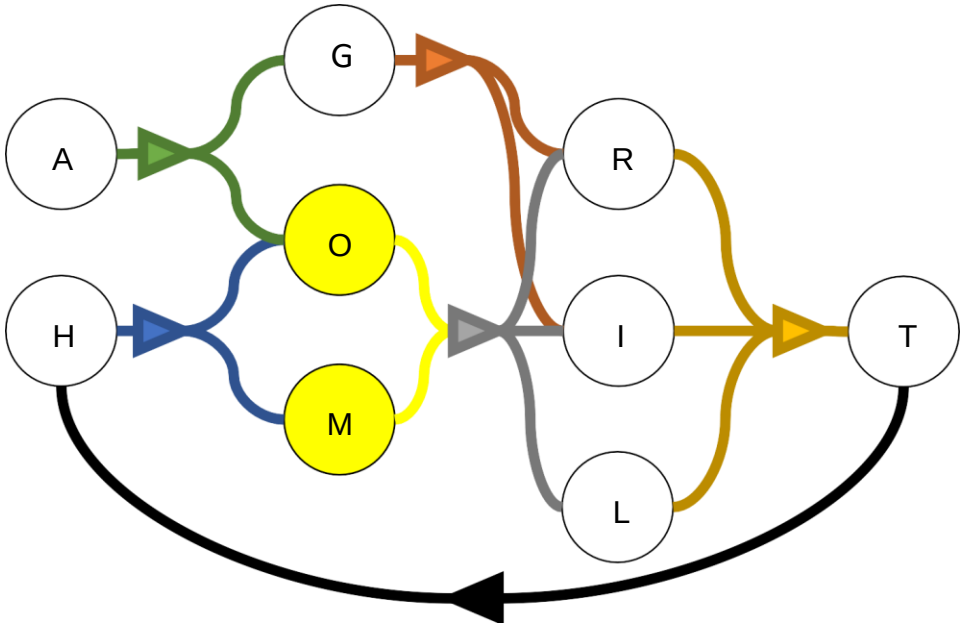
Visualization of FDs: directed hypergraphs



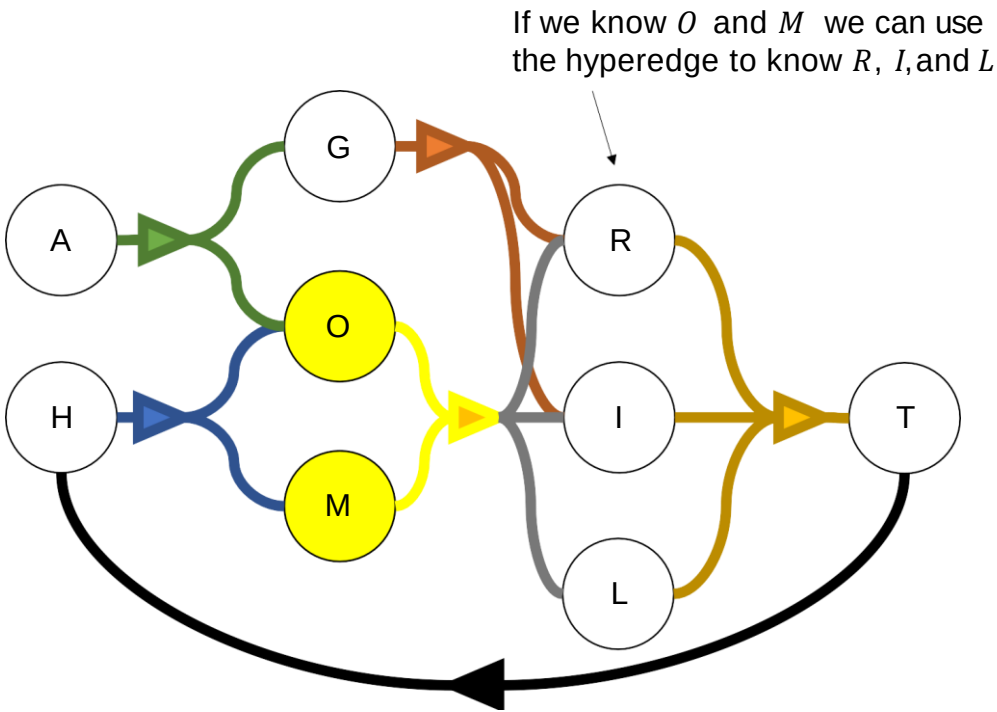
Visualization of FDs: directed hypergraphs



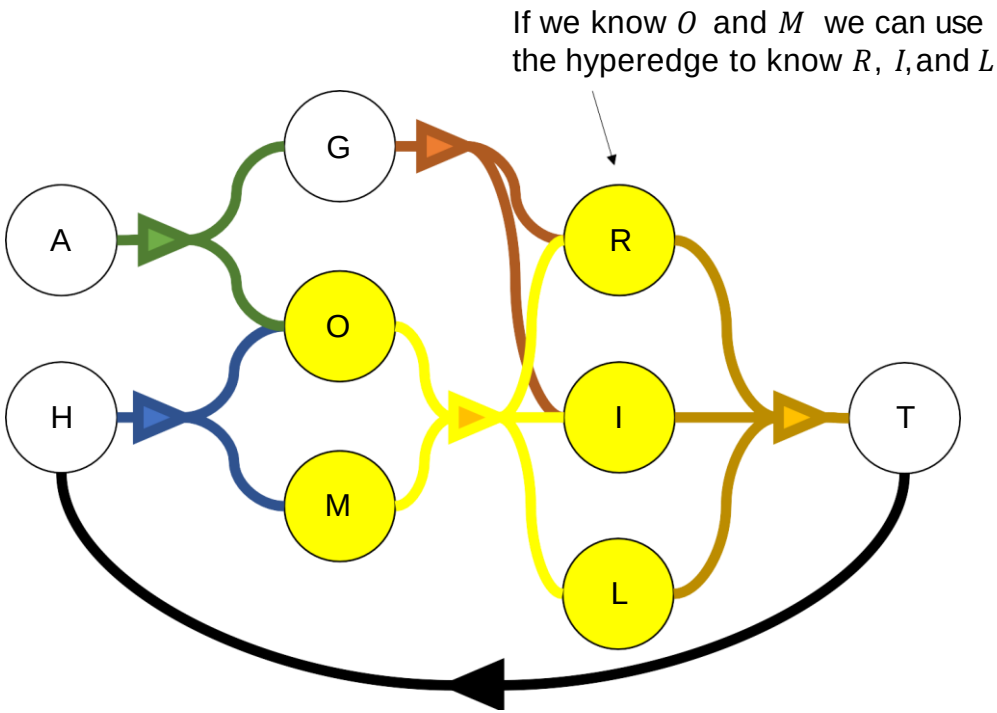
Visualization of FDs: directed hypergraphs



Visualization of FDs: directed hypergraphs

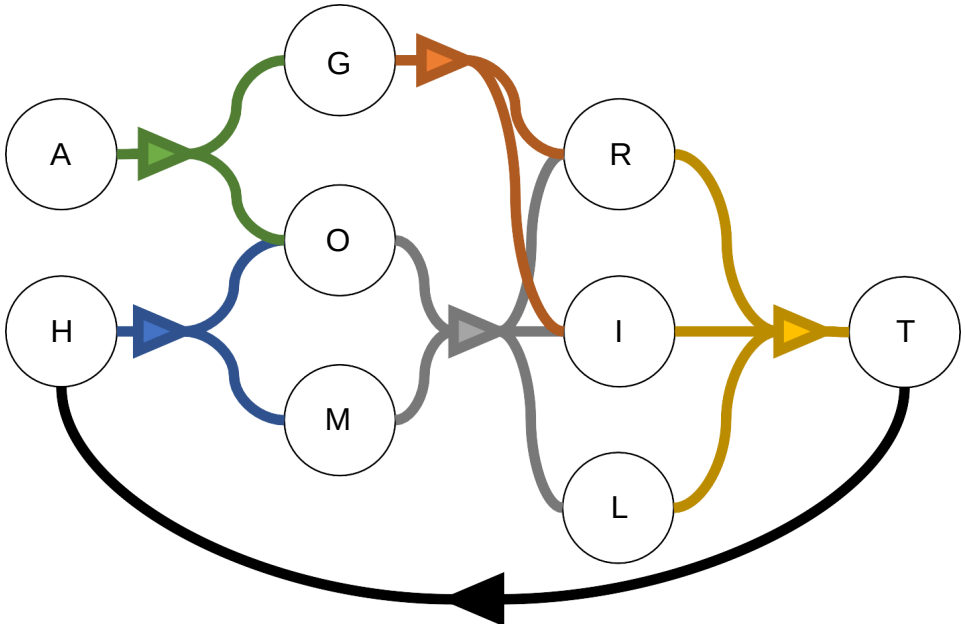


Visualization of FDs: directed hypergraphs



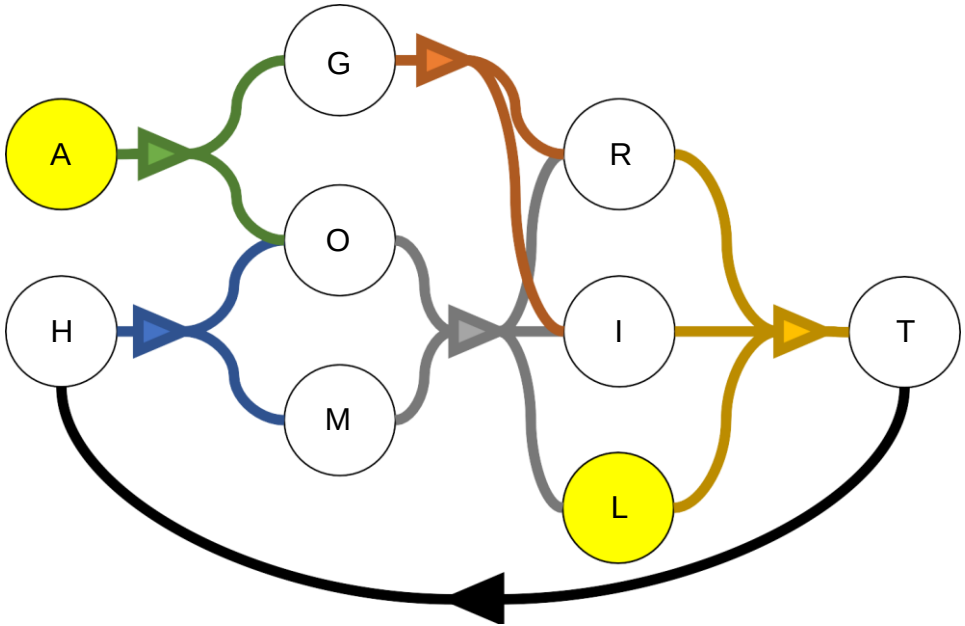
Visualization of FDs: directed hypergraphs

Compute the closure $\{A,L\}^+$



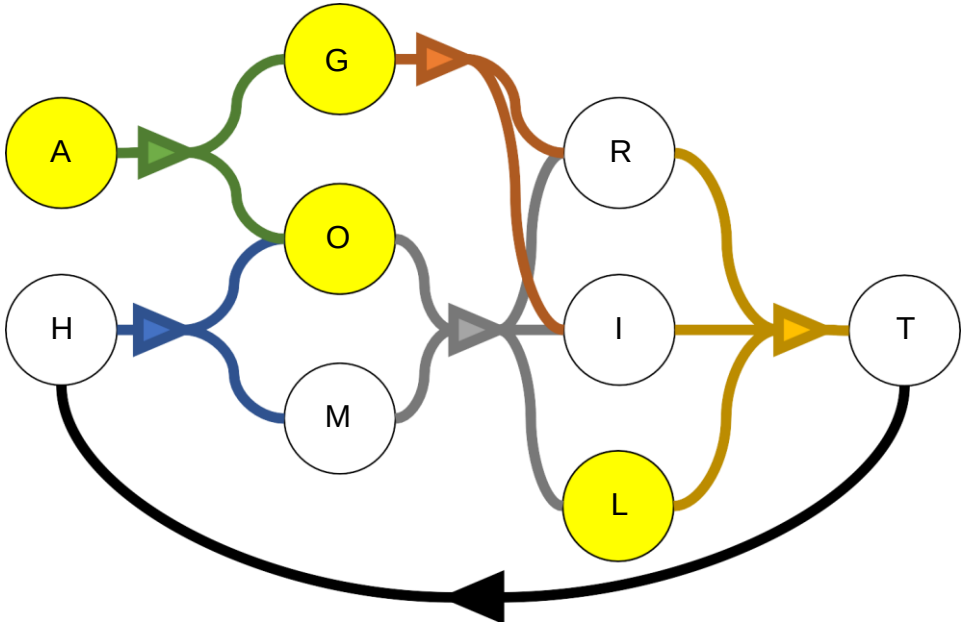
Visualization of FDs: directed hypergraphs

$$\{A,L\}^+ = \{A, L, \ldots\}$$



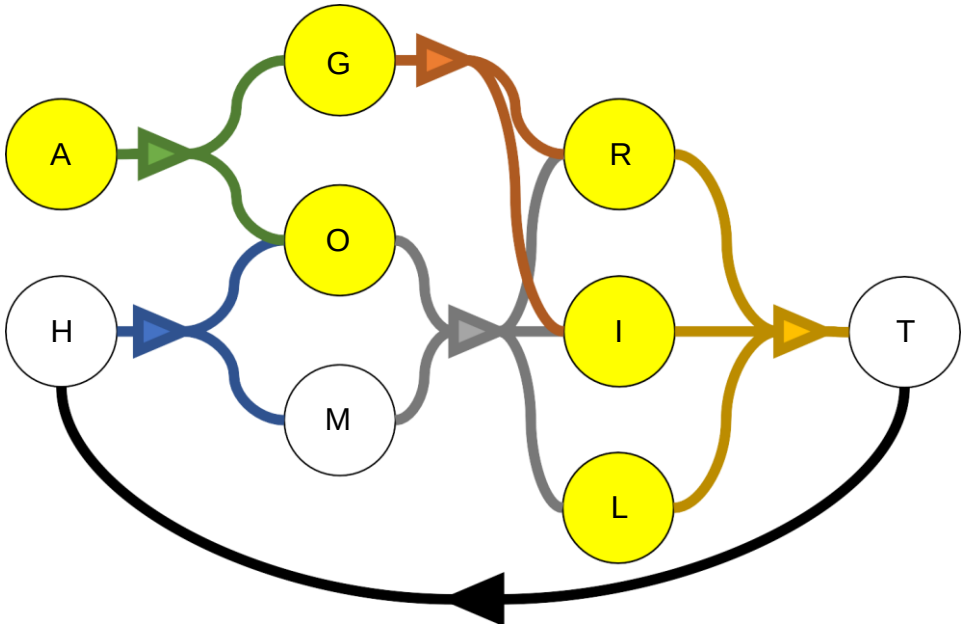
Visualization of FDs: directed hypergraphs

$$\{A, L\}^{+} = \{A, L, G, O, \ldots\}$$



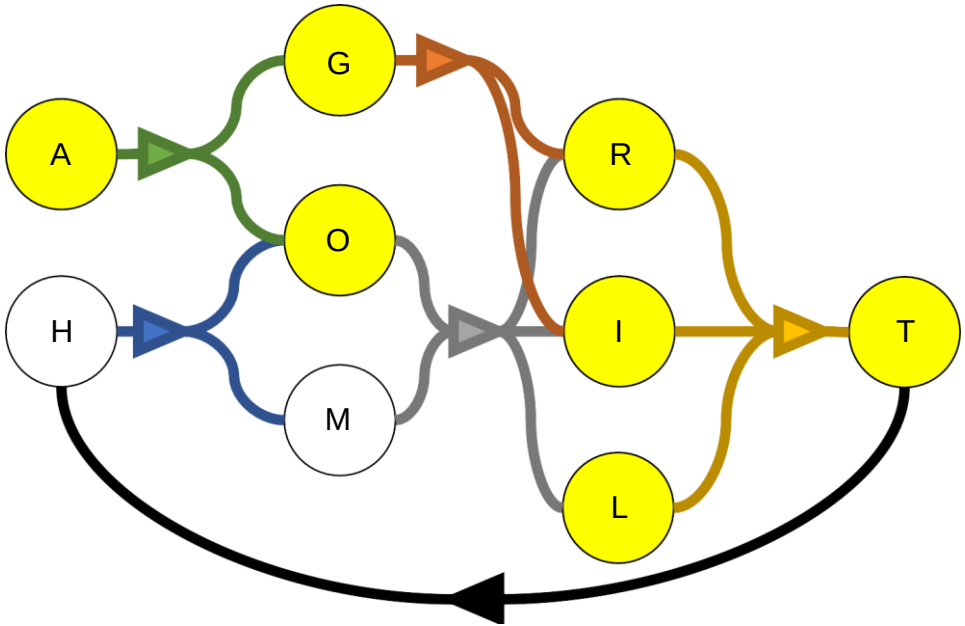
Visualization of FDs: directed hypergraphs

$$\{A, L\}^+ = \{A, L, G, O, R, I \ldots\}$$



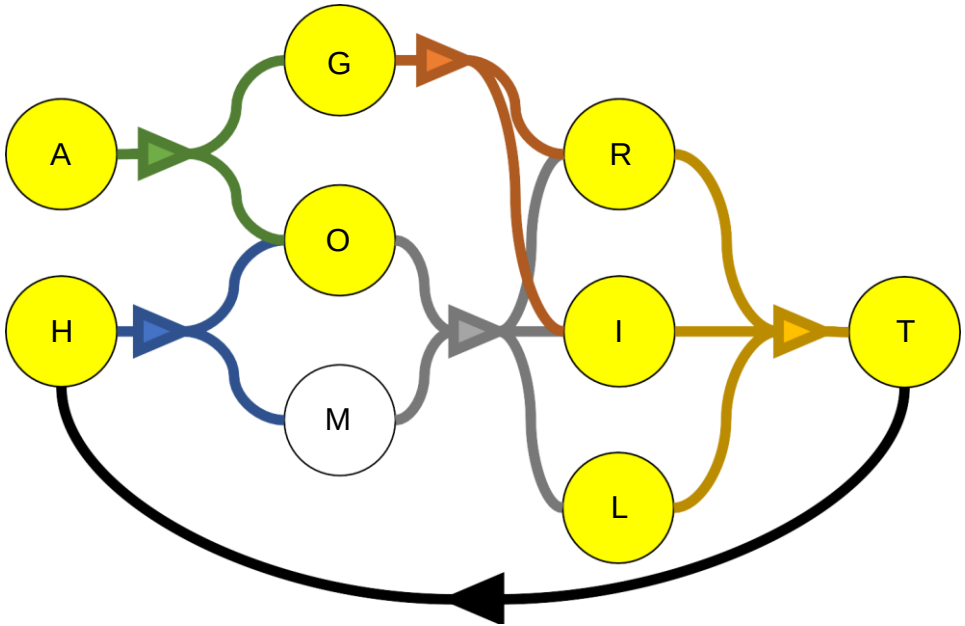
Visualization of FDs: directed hypergraphs

$$\{A, L\}^+ = \{A, L, G, O, R, I, T \ldots\}$$



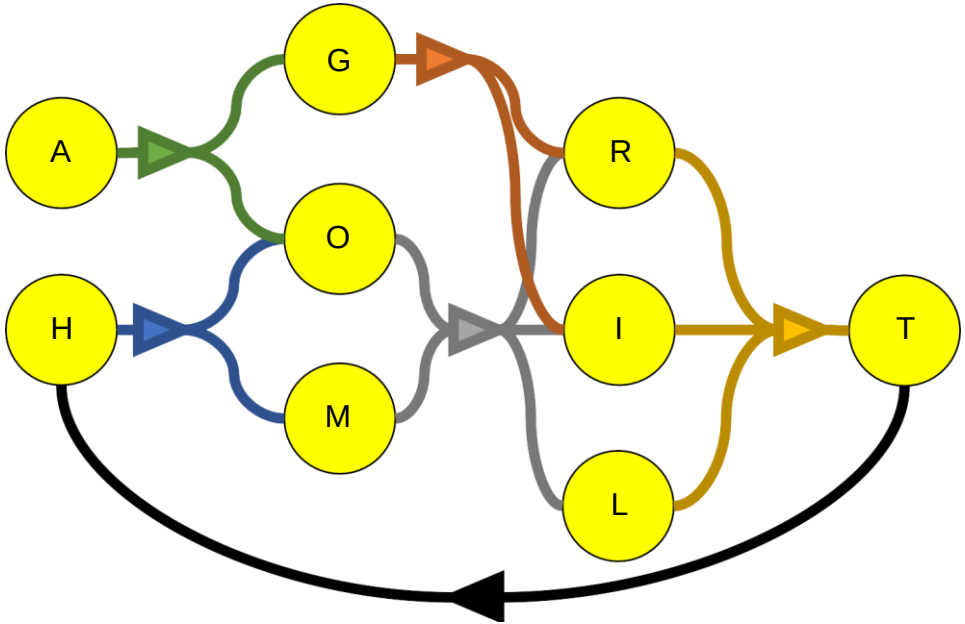
Visualization of FDs: directed hypergraphs

$$\{A, L\}^+ = \{A, L, G, O, R, I, T, H \ \ldots\}$$



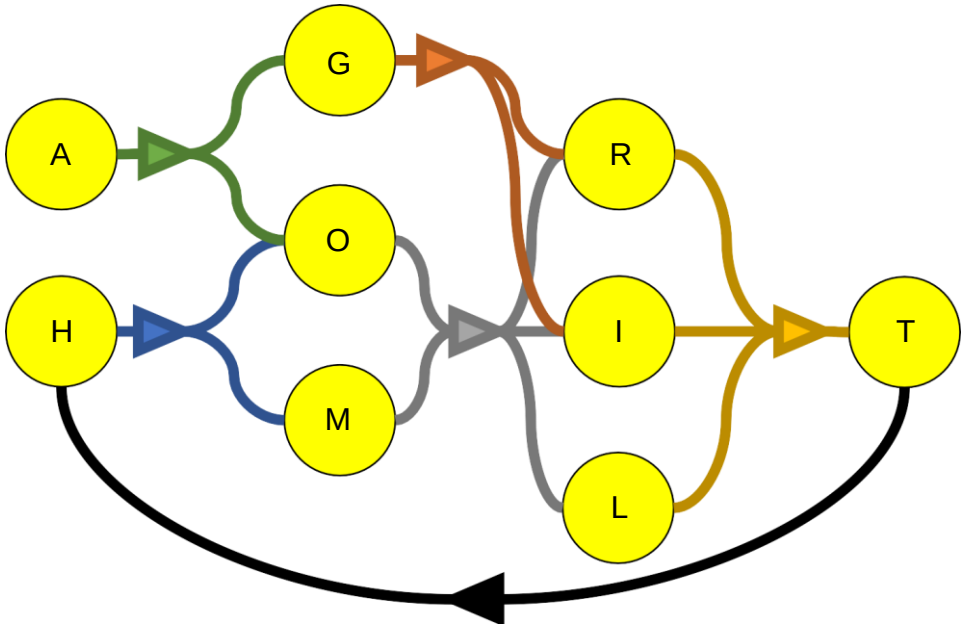
Visualization of FDs: directed hypergraphs

$$\{A, L\}^+ = \{A, L, G, O, R, I, T, H, M\}$$



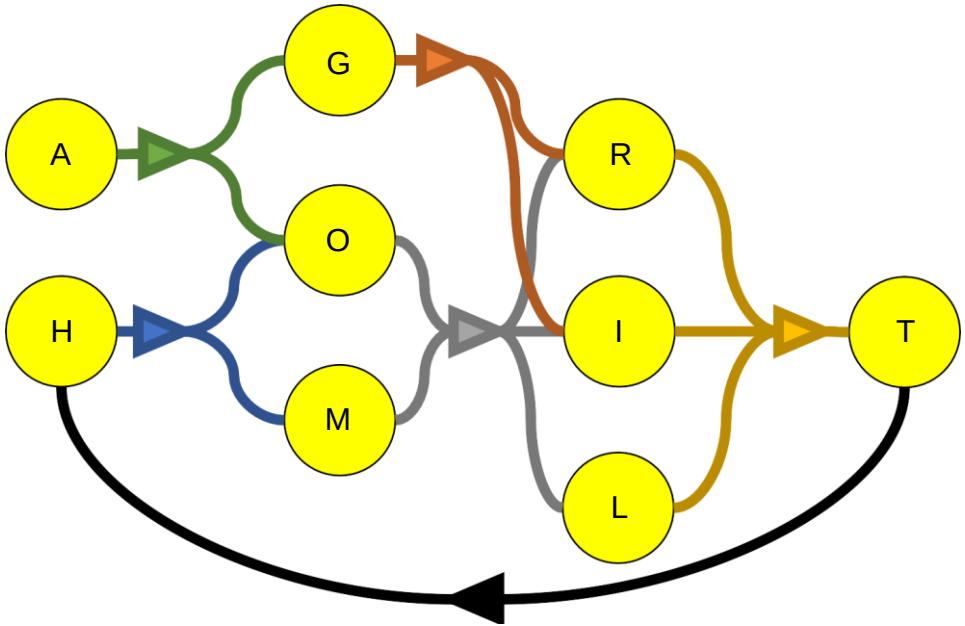
Visualization of FDs: directed hypergraphs

$\{A, L\}$ determines all attributes



Visualization of FDs: directed hypergraphs

$\{A, L\}$ is a (super)key!



What is normalization?

Normalization is a technique for organizing the data into multiple related tables, to “minimize” data redundancy.

What is data redundancy?

sid	name	pillar	hod	office_tel
123	Agus	ISTD	Mr. Tan	53337
456	Bron	ISTD	Mr. Tan	53337
789	Hannah	ISTD	Mr. Tan	53337
012	Dewi	ISTD	Mr. Tan	53337

Repetition of the same data at multiple places

Why to reduce it?

- Increases the size of the database
- Insert, delete, update problems

Types of normalization

Normalization can be achieved in several “forms”:

- 1st normal form (1NF)
- 2nd normal form (2NF)
- **3rd normal form (3NF)**
- **Boyce-Codd normal form (BCNF)**
- 4th, 5th, 6th normal forms (4NF, 5NF, 6NF)…

1st normal form

1st step of the normalization process
1NF expects that the table is designed such that it can be easily extended.
1NF is mandatory!

How to achieve 1NF?

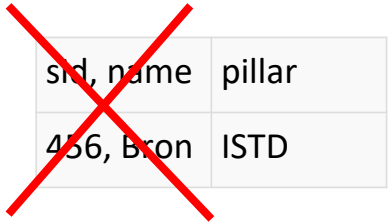
4 basic rules that a table should follow to be in 1NF:

Each column should contain atomic values

A column should contain values of the same type

Each column should have a unique name

Order in which data is saved does not matter.



sid, name	pillar
456, Bron	ISTD

1NF: formal definition

1NF

A relation R is in **First Normal Form** if all attribute values are atomic. Attribute values cannot be multivalued.

We call data in 1NF "flat."

How to achieve 1NF?

sid	name	subject
123	Agus	Database, Computing
456	Bron	Database
789	Hannah	Architecture, Mathematics

Violates 1NF

Split

sid	name	subject
123	Agus	Database
123	Agus	Computing
456	Bron	Database
789	Hannah	Architecture
789	Hannah	Mathematics

1NF: conclusion

Some values are repeated but values for each column are now atomic for each record/row.

Using 1NF, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

2nd Normal Form (2NF)

2 conditions for a table to be in 2NF:

- The table has to be in 1NF.
- It should not have Partial Dependencies.

What is Partial Dependency?

Definition – Full Functional Dependency

In the functional dependency $X \rightarrow A$, an attribute A is **Fully Functionally Dependent** on X if there is no subset Y of X in which $Y \rightarrow A$ holds.

Otherwise, if there is some Y where $Y \rightarrow A$ holds, A is **Partially Dependent** on X.

e.g. **grade** is fully functionally dependent on **sid,course**

2NF: Formal definition

Definition – Second Normal Form (2NF)

A relation R is in **Second Normal Form** if it is in 1NF and all nonprime attributes are fully functionally dependent on the primary key of R.

- **Definition – Prime Attribute**
- An attribute is a **Prime Attribute** if it is part of a candidate key, otherwise the attribute is considered **Nonprime**.

Some recall on keys

Given a super/candidate/primary key, you can fetch any row of data in a table.

student(sid,name,address)

sid	name	address	phone
123	Agus	Bedok	9191
111	Agus	Tampines	2390

{sid,name} is a superkey.
sid and **phone** are candidate keys.
sid is more likely to be the primary key.
If the key is multivalued, we say that the key is composite.

Definition – Prime Attribute

An attribute is a **Prime Attribute** if it is part of a candidate key, otherwise the attribute is considered **Nonprime**

Some recall on keys

Given a primary key, you can fetch any row of data in a table.

student(sid,name,address,phone)

sid	name	address	phone
123	Agus	Bedok	9191
111	Agus	Tampines	2390

subject(sub_id,sub_name)

sub_id	sub_name
1	Java
2	English
3	Matlab

score(sid,sub_id,marks,teacher)

sid	sub_id	marks	teacher
123	1	70	Oka
123	2	35	Chris
111	1	80	Oka

{sid,sub_id} forms a candidate key.
We choose it as our primary key.

Let’s go back to partial dependencies…

score(sid,sub_id,marks,teacher)

sid	sub_id	marks	teacher
123	1	70	Oka
123	2	35	Chris
111	1	80	Oka

{sid,sub_id} forms a candidate key. We choose it as our primary key.

- In the score table, teacher only depends on the subject, not on the student id.
- Partial dependency: an attribute in a table depends on only a part of the primary key and not on the whole key!
- The score table is not in 2NF.

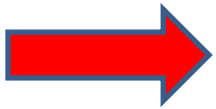
How to get rid of partial dependencies?

subject(sub_id,sub_name)

sub_id	sub_name
1	Java
2	English
3	Matlab

score(sid,sub_id,marks,teacher)

sid	sub_id	marks	teacher
123	1	70	Oka
123	2	35	Chris
111	1	80	Oka



subject(sub_id,sub_name,teacher)

sub_id	sub_name	teacher
1	Java	Oka
2	English	Chris
3	Matlab	Anh

score(sid,sub_id,marks)

sid	sub_id	marks
123	1	70
123	2	35
111	1	80

2NF: conclusion

2NF = 1NF + No Partial Dependency.
Partial Dependency exists when, for a composite primary key, any attribute in the table depends only on a part of the primary key (i.e. not on the complete primary key).
To get rid of Partial dependency, divide the table, remove the attribute which is causing partial dependency, and move it to some other table where it fits in well.

3rd normal form (3NF)

2 conditions for a table to be in 3NF:

The table has to be in 2NF.

It should not have Transitive Dependencies.

3NF: formal definition

Definition – Third Normal Form (3NF)

A relation R is in **Third Normal Form** if it is in 2NF and if for all non-trivial FDs, $X \rightarrow A$, X is a superkey or A contains only prime attributes

What is transitive dependency?

score(sid,sub_id,marks,exam_type,total)

sid	sub_id	marks	exam_type	total
123	1	70	main	100
123	2	35	oral	40
111	1	80	main	100
111	2	81	main	100

- Our primary key is **{sid,sub_id}.**
- Here, exam_type depends on the subject and on student's pillar: so, depends on the primary key.
- However, total depends on exam_type. There is transitive dependency!
- **{sid,sub_id}** → **exam_type** → **total**
- The score table is not in 3NF!

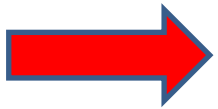
How to remove transitive dependency?

Let **X** be a primary key and **X** → **Y** → **Z** be a transitive dependency.
A solution is (1) to remove Y and Z from the main table,
(2) to create a subtable with Y and Z
(3) to add an attribute A in both tables to join them.

How to remove transitive dependencies?

score(sid,sub_id,marks,exam_type,total)

sid	sub_id	marks	exam_type	total
123	1	70	main	100
123	2	35	oral	40
111	1	80	main	100
111	2	81	main	100



score(sid,sub_id,marks,eid)

sid	sub_id	marks	eid
123	1	70	1
123	2	35	2
111	1	80	1
111	2	81	1

exam(eid,exam_type,total)

eid	exam_type	total
1	main	100
2	oral	40

3NF: conclusion

3NF = 2NF + No Transitive Dependency.
Transitive Dependency exists when an attribute in the table does not depend directly from the primary key but from intermediate attributes.
To get rid of Transitive dependency: remove the attributes which are causing transitive dependency in the main table, create a subtable, add an attribute to join both tables.
Removing transitive dependency:

- Amount of data duplication is reduced.
- Data integrity achieved.

Boyce-Codd normal form (BCNF)

Upgraded version of 3NF. Sometimes called 3.5 normal form.

2 conditions for a table to be in BCNF:

The table should be in 3NF.

For any dependency **X** → **Y**, **X** should be a superkey.

BCNF: formal definition

BCNF

A relation R is in **Boyce-Codd Normal Form (BCNF)** if for every non-trivial dependency, $X \rightarrow A$, X is a superkey.

Equivalently, a relation R is in BCNF if $\forall X$ either $X^+ = X$ or $X^+ = C$ where C is the set of all attributes in R

Examples

- $R(A, B, C)$ with FDs $A \rightarrow B$ and $B \rightarrow C$ BCNF?
- $R(A, B, C)$ with FDs $A \rightarrow BC$ BCNF?
- $R(A, B, C)$ and $S(A, D, E)$ with FDs $A \rightarrow BCDE$ and $E \rightarrow AD$ BCNF?

BCNF: formal definition

BCNF

A relation R is in **Boyce-Codd Normal Form (BCNF)** if for every non-trivial dependency, $X \rightarrow A$, X is a superkey.

Equivalently, a relation R is in BCNF if $\forall X$ either $X^+ = X$ or $X^+ = C$ where C is the set of all attributes in R

Examples

- $R(A, B, C)$ with FDs $A \rightarrow B$ and $B \rightarrow C$ is not in BCNF
- $R(A, B, C)$ with FDs $A \rightarrow BC$ is in BCNF
- $R(A, B, C)$ and $S(A, D, E)$ with FDs $A \rightarrow BCDE$ and $E \rightarrow AD$ is in BCNF

Recall on dependencies

Attribute → **attribute** — Functional dependency

Part of primary key → **non-prime attribute** — Partial dependency
Not allowed in 2NF

Non-prime attribute → **non-prime attribute** — Transitive dependency
Not allowed in 3NF

Attribute not part of the primary key → **Attribute from the primary key**

Is this possible? Not in BCNF.

Can an attribute, not in the primary key, can determine an attribute from the primary key ?

subject(sid,subject_name,teacher)

sid	subject_name	teacher
123	Java	Oka
123	English	Chris
456	Java	Norman
789	Maths	Cyrille
012	Java	Oka

We assume that:

1. Only one course is assigned to teachers.
2. Depending on the student's pillar, the teacher may be different for the same subject.

- Multiple professors teach Java
- sid,subject_name is our primary key: **sid,subject_name** → **teacher**
- But we also have **teacher** → **subject_name**

1-2-3-BC-NF

We assume that:

1. Only one course is assigned to teachers.
2. Depending on the student's pillar, the teacher may be different for the same subject.

subject(sid,subject_name,teacher)

sid	subject_name	teacher
123	Java	Oka
123	English	Chris
456	Java	Norman
789	Maths	Cyrille
012	Java	Oka

- All attributes are single-valued: satisfies 1NF.
- We have **sid,subject_name** → **teacher**. But we don't have **sid** → **teacher** or **subject_name** → **teacher**. No partial dependencies: satisfies 2NF.
- No transitive dependencies as well. Indeed, even though teacher is not part of our primary key, it is also a prime attribute because {sid, teacher} is another candidate key: satisfies 3NF.
- But because of $\{teacher\}^+ = \{teacher,subject\_name\}$, the table does not satisfy BCNF.

How to make the table satisfy BCNF?

subject(sid,subject_name,teacher)

sid	subject_name	teacher
123	Java	Oka
123	English	Chris
456	Java	Norman
789	Maths	Cyrille
012	Java	Oka

student(sid,tid)

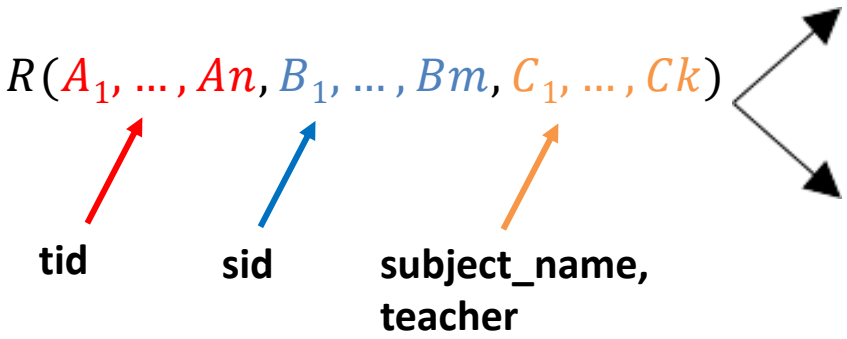
sid	tid
123	1
123	2
456	3
789	4
012	1

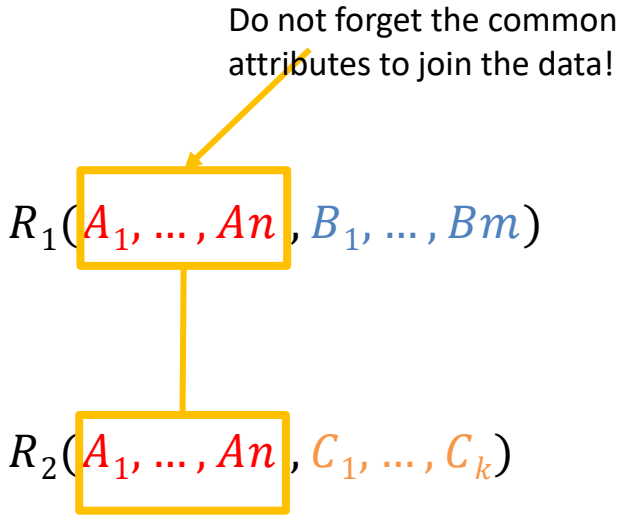
teacher(tid,subject_name,teacher)

tid	subject_name	teacher
1	Java	Oka
2	English	Chris
3	Java	Norman
4	Maths	Cyrille

Decomposition

- **Extract** attributes using **decomposition** (split the schema into smaller parts) based on Heath's theorem (cf Annex slide 128).
- Here, decomposition means:





BCNF decomposition algorithm

$Normalize(R)$

$C \leftarrow$ the set of all attributes in R **find** X **s.t.** $X^+ \neq X$ **and** $X^+ \neq C$ **if** X is not found

then "R is in BCNF"

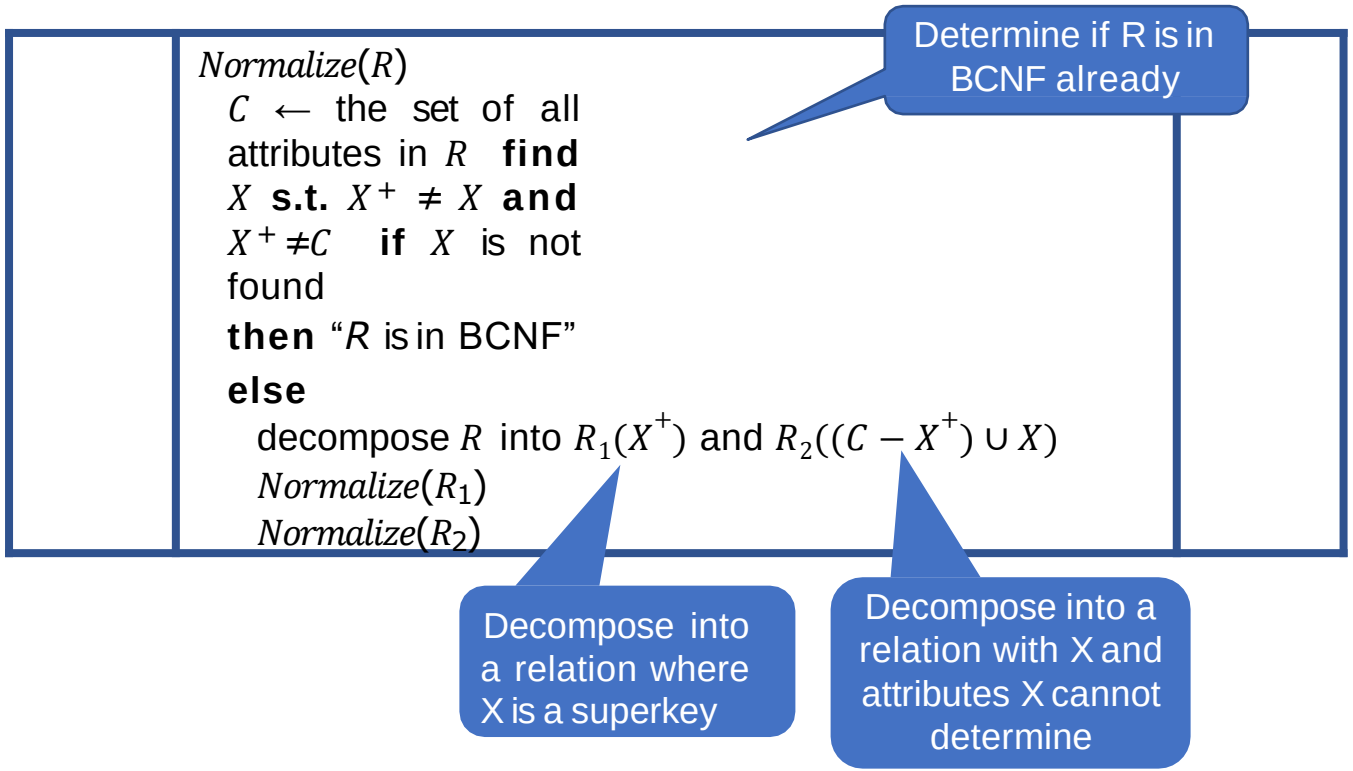
else

decompose R into $R_1(X^+)$ and $R_2((C - X^+) \cup X)$

$Normalize(R_1)$

$Normalize(R_2)$

BCNF decomposition algorithm



Losslessness

Definition

Lossless Decomposition is a reversible decomposition, i.e. rejoining all decomposed relations will always result exactly with the original data.

This is the opposite of a **Lossy Decomposition**, an irreversible decomposition, where rejoining all decomposed relations may result something other than the original data, specifically with extra tuples.

BCNF: pros and cons

- **Is BCNF decomposition lossless? Yes!**
  - For those who are interested: look at **Heath's theorem (see slide 65).**

- **Does BCNF preserve the FDs? Not necessarily.**
  - Some FDs of the original relation may not be all covered after the decomposition.

BCNF: conclusion

BCNF = 3NF + No dependency of type non-prime → prime.
To get rid of non-prime → prime : use the BCNF algorithm.
BCNF decomposition is lossless but does not necessarily preserve the dependencies.

4th normal form (4NF)

2 conditions for a table to satisfy 4NF:

It should satisfy BCNF.

It should not have multi-valued dependency.

What is multi-valued dependency?

Definition

If the table has at least 3 columns (**A**,**B**,**C**),
If, given a functional dependency **A** → **B**, for a single value A_1 of attribute **A**, more than one value (e.g. B_1 and B_2) of **B** exist,
And if **B** and **C** are independent of each other,
the table has a multi-valued dependency.

Multi-valued dependency

student(sid,subject,hobby)

sid	subject	hobby
123	Java	chess
123	English	wine tasting
456	Java	tennis
789	Maths	wushu
012	Java	salsa

Multi-valued dependency leads to unnecessary repetition of data.

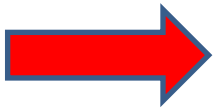
student(sid,subject,hobby)

sid	subject	hobby
123	Java	chess
123	Java	wine tasting
123	English	chess
123	English	wine tasting
456	Java	tennis
789	Maths	wushu
012	Java	salsa

How to remove multi-valued dependency?

student(sid,subject,hobby)

sid	subject	hobby
123	Java	chess
123	English	wine tasting
456	Java	tennis
789	Maths	wushu
012	Java	salsa



student(sid,subject)

sid	subject
123	Java
123	English
456	Java
789	Maths
012	Java

student(sid,hobby)

sid	hobby
123	chess
123	wine tasting
456	tennis
789	wushu
012	salsa

4NF and more: conclusion

4NF = BCNF + No multi-valued dependency
Multi-valued dependency can be easily removed by “separating” the independent multi-valued attributes into subtables.
There exist 5NF and 6NF but they are not necessary (and may even affect performance).
3NF and/or BCNF are the main standards of database. You should always decompose your database in 3NF or even in BCNF!

Is a decomposition lossless?

- We need a way to verify if a decomposition is lossless.
- That means to check that joining decompositions S_1,…,S_n equals the original relation R:

$$R = S_1 \bowtie \cdots \bowtie Sn \ ?$$

- Showing $R \subseteq S_1 \bowtie \cdots \bowtie Sn$ is usually simple. Just check if you can naturally join the subtables (need of a joint attribute) and if all the attributes of R are represented.
- Showing $R \supseteq S_1 \bowtie \cdots \bowtie Sn$ is trickier.

Chase method (step-by-step)

You can determine with a chase algorithm if a decomposition is lossless.

	1.Generate a tableau of generic tuples (a,b,c,d) representing each schema. 2. Each generic tuple (a,b,c,d) has known values corresponding to the respective projection. 3.Until a row reflects the original generic tuple, continue to chase on FDs (extract more agreements of values)	

Chase method: an example

Let R(A,B,C,D) be a relation with FDs: **A** → **B**, **B** → **C**, **CD** → **A** and S_1(A,D), S_2(A,C) and S_3(B,C,D) be a decomposition of R into 3 (projected) relations.
We want to prove $R \supseteq S_1 \bowtie S_2 \bowtie S_3$.
Let's prove that $(a, b, c, d) \in S_1 \bowtie S_2 \bowtie S_3$ implies $(a, b, c, d) \in R$.
We already know that $(a, d) \in S_1$, $(a, c) \in S_2$ and $(b, c, d) \in S_3$.

1st line corresponds to S1
2nd line to S2, 3rd line to S3

A	B	C	D
a	b1	c1	**d**
a	b2	**c**	d2
a3	**b**	**c**	**d**

Chase method: an example

Let R(A,B,C,D) be a relation with FDs: **A** → **B**, **B** → **C**, **CD** → **A** and S_1(A,D), S_2(A,C) and S_3(B,C,D) be a decomposition of R into 3 (projected) relations.
The tableau can be chased by applying the FDs to equate symbols in the tableau. Final tableau with a row that is the same as *(a,b,c,d)* implies that any tuple *(a,b,c,d)* in the join of the projections is a tuple of *R.*

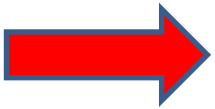
A	B	C	D
a	b1	c1	**d**
a	b2	**c**	d2
a3	**b**	**c**	**d**

Apply **A** → **B**

A	B	C	D
a	b1	c1	**d**
a	b1	**c**	d2
a3	**b**	**c**	**d**

When equating two symbols, if both have their own subscript, uniformize

Apply **B** → **C**



A	B	C	D
a	b1	**c**	**d**
a	b1	**c**	d2
a3	**b**	**c**	**d**

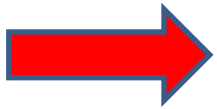
When equating two symbols, if one of them is unsubscripted, make the other be the same.

Chase method: an example

A	B	C	D
a	b1	c	d
a	b1	c	d2
a3	b	c	d

Apply **CD** → **A**

A	B	C	D
a	b1	c	d
a	b1	c	d2
a	b	c	d



$(a, b, c, d) \in R$

When equating two symbols, if one of them is unsubscripted, make the other be the same.

Conclusion

What have you seen today?

- Recall about keys, functional dependencies…
- Normal forms
- Losslessness
- Chase method

Annex: Heath's Theorem

Definition – Heath's Theorem

Suppose we have the relation R and three disjoint subsets of the attributes of R we will write as $A_1, \ldots, A_n$, $B_1, \ldots, B_m$, and $C_1, \ldots, C_k$. Suppose we also have a FD that is $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$.

Heath's Theorem states that the decomposition of R into $R_1(A_1, \ldots, A_n, B_1, \ldots, B_m)$ and $R_2(A_1, \ldots, A_n, C_1, \ldots, C_k)$ is lossless where R_1 and R_2 are the projections of R on their respective attributes.

$$R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_k) \begin{cases} \rightarrow R_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \\ \rightarrow R_2(A_1, \ldots, A_n, C_1, \ldots, C_k) \end{cases}$$

By reflection, the same decomposition of R under the alternate FD $A_1, \ldots, A_n \rightarrow C_1, \ldots, C_k$ is also lossless.