Functional Dependency & Normalization

- Use decomposition to improve database design

- Anomalies arise when we combine too much into one relation
- These anomalies are caused by certain functional dependencies
- Decomposition can help remove functional dependencies
- If a decomposed relation does not have certain dependencies, it is said to be in some normal form
- Normalization is the process of decomposing relations into normal forms so as to eliminate anomalies

Anomalies

A poorly designed database causes ***anomalies***:

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

If every course is in only one room, contains ***redundant*** information!

(Student, Course) $\longrightarrow$ Room is a ***partial dependency***.
Partial dependencies can cause redundancy.

Anomalies

A poorly designed database causes ***anomalies***:

Student	Course	Room
Mary	CS145	B01
Joe	CS145	$\boxed{\text{C12}}$
Sam	CS145	B01
..	..	..

If we update the room number for one tuple, we get inconsistent data = an ***<u>update</u>*** **<u>anomaly</u>**

Redundancy causes other anomalies.

Anomalies

A poorly designed database causes ***anomalies***:

Student	Course	Room
..	..	..

If everyone drops the class, we lose what room the class is in! = a ***delete* anomaly**

...	CS229	C12

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

Similarly, we can't reserve a room without students = an ***insert* anomaly**

Decomposition

Student	Course
Mary	CS145
Joe	CS145
Sam	CS145
..	..

Course	Room
CS145	B01
CS229	C12

Is this form better?

- Redundancy?
- Update anomaly?
- Delete anomaly?
- Insert anomaly?

Functional Dependency

Def: Let A,B be *sets* of attributes
We write $A \longrightarrow B$ or say A ***functionally determines*** B if, for any tuples t_1 and t_2:

$$t_1[A] = t_2[A] \text{ implies } t_1[B] = t_2[B]$$

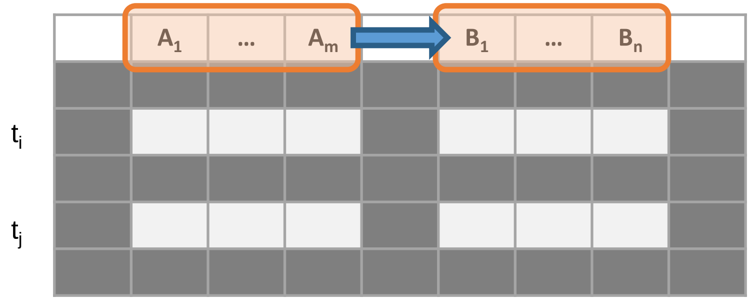
and we call $A \longrightarrow B$ a **functional dependency**

$A \longrightarrow B$ means that
"whenever two tuples agree on A then they agree on B".

Functional Dependency

<u>Def (again):</u>
Given attribute sets **A = {A$_1$,…,A$_m$}** and **B = {B$_1$,…,B$_n$}** in **R,**

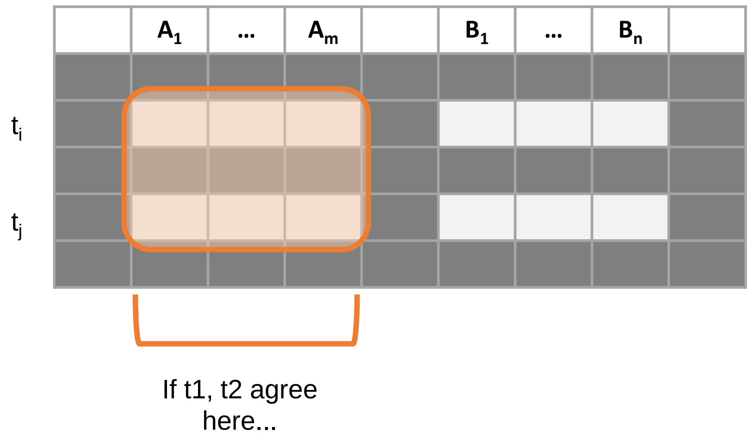
Functional Dependency



Def (again):
Given attribute sets $\mathbf{A = \{A_1,\ldots,A_m\}}$ and $\mathbf{B = \{B_1,\ldots,B_n\}}$ in **R,**

The ***functional dependency*** **A ⟶ B on R** holds if for ***any*** t_i,t_j in R:

Functional Dependency

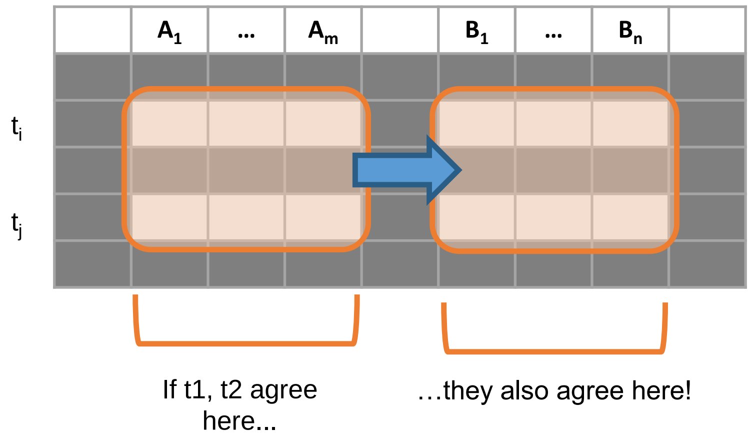


<u>Def (again):</u>
Given attribute sets **A = {A_1,…,A_m}** and **B = {B_1,…,B_n}** in **R,**

The ***functional dependency*** **A ⟶ B on R** holds if for ***any*** t_i,t_j in R:

<u>if</u> $t_i[A_1] = t_j[A_1]$ AND $t_i[A_2] = t_j[A_2]$ AND … AND $t_i[A_m] = t_j[A_m]$

Functional Dependency



Def (again):
Given attribute sets $\mathbf{A = \{A_1,\ldots,A_m\}}$ and $\mathbf{B = \{B_1,\ldots,B_n\}}$ in **R,**

The ***functional dependency*** $\mathbf{A \longrightarrow B}$ **on R** holds if for ***any*** t_i,t_j in R:

if $t_i[A_1] = t_j[A_1]$ AND $t_i[A_2] = t_j[A_2]$ AND … AND $t_i[A_m] = t_j[A_m]$

then $t_i[B_1] = t_j[B_1]$ AND $t_i[B_2] = t_j[B_2]$ AND … AND $t_i[B_n] = t_j[B_n]$

Keys & Superkeys

A **superkey** is a set of attributes $\mathbf{A_1, \ldots, A_n}$ s.t.
for *any other* attribute **B** in R,
we have $\{\mathbf{A_1, \ldots, A_n}\} \longrightarrow \mathbf{B}$

I.e. all attributes are *functionally determined* by a superkey

A **key** is a *minimal* superkey

This means that no proper subset of a key is also a superkey (i.e., dropping any attribute from the key makes it no longer a superkey)

The entire set of attributes in R is always a superkey.

Full/Partial Dependency

- A functional dependency $X \longrightarrow Y$ is a ***partial dependency*** if for some $A \in X$, $(X - \{A\}) \longrightarrow Y$

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

If every course is in only one room, contains ***redundant*** information!

(Student, Course) $\longrightarrow$ Room is a ***partial dependency***.
Partial dependencies can cause redundancy.

Second Normal Form

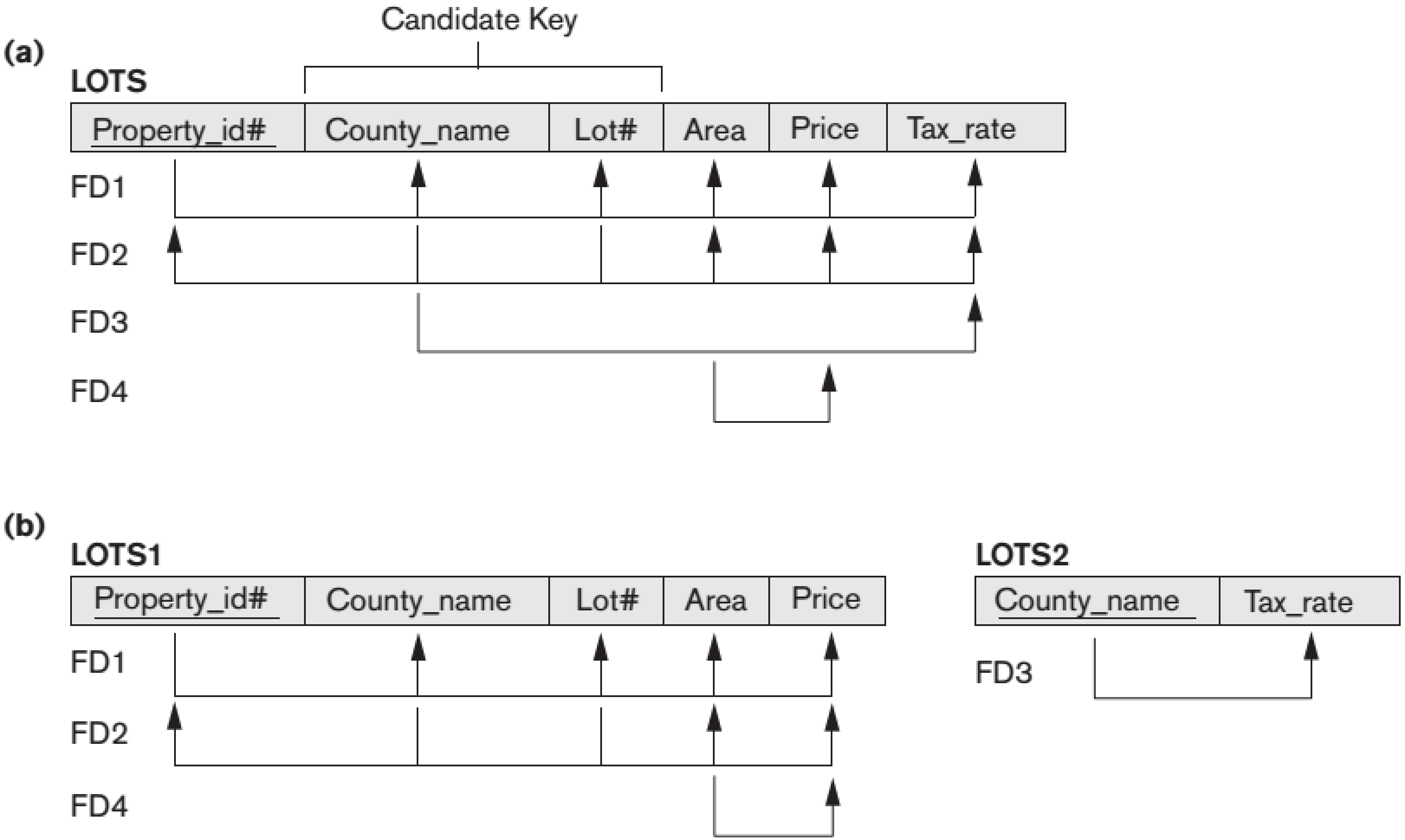
- A relation schema R is in second normal form (2NF) if there is ***no partial dependency*** from any (composite) key of R to any non-key attribute in R

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

Student	Course
Mary	CS145
Joe	CS145
Sam	CS145
..	..

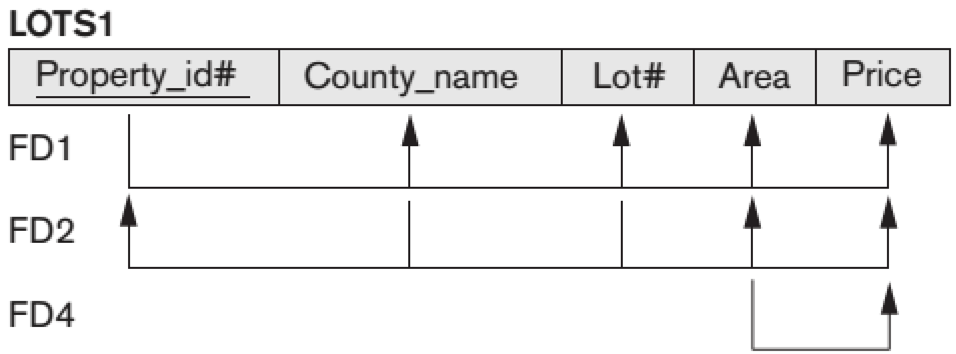
Course	Room
CS145	B01
CS229	C12

Second Normal Form



Transitive Dependency

- A functional dependency $X \longrightarrow Y$ in a relation schema R is a ***transitive dependency*** if there exists a set of attributes Z in R such that both $X \longrightarrow Z$ and $Z \longrightarrow Y$ hold and that Z is neither a key nor a subset of any key of R

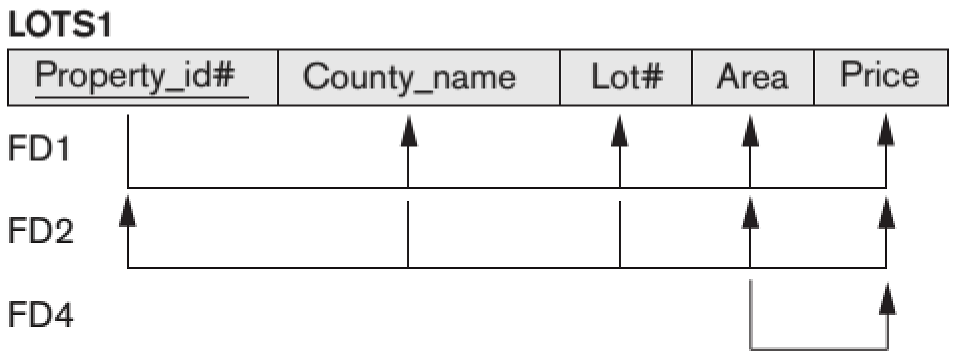


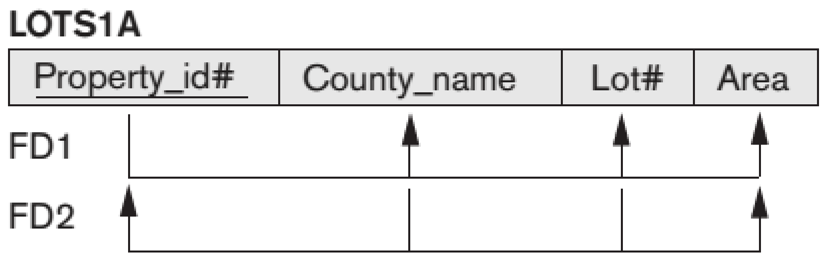
delete anomaly

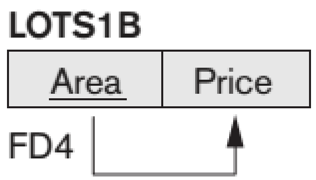
insert anomaly

Third Normal Form

- A relation schema R is in third normal form (3NF) if there is ***no partial dependency or transitive dependency*** from any key of R to any non-key attribute in R





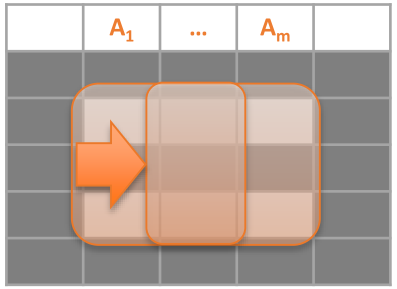


Finding Implicit Functional Dependencies

- Given a set of FD’s, what other FD’s must hold?

- Armstrong’s axioms
  - Reflexivity
  - Augmentation
  - Transitivity
- The splitting/combining rule

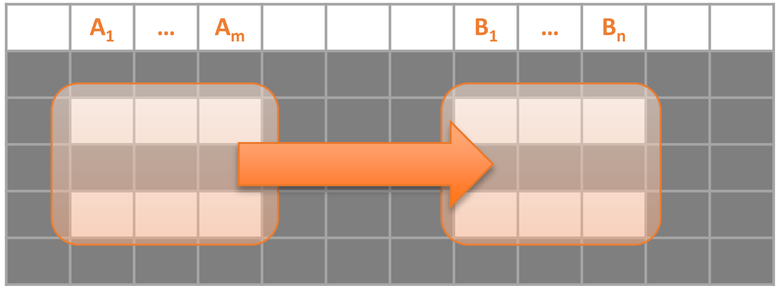
Reflexivity

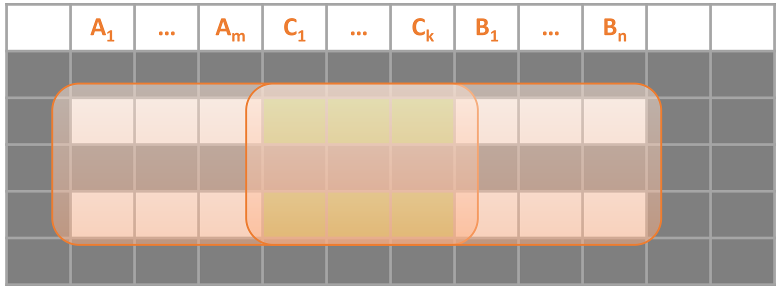


$A_1,\dots,A_m \longrightarrow B_1,\dots,B_n$ if $\{B_1,\dots,B_n\}$ is a subset of $\{A_1,\dots,A_m\}$

These are called trivial FD's

Augmentation

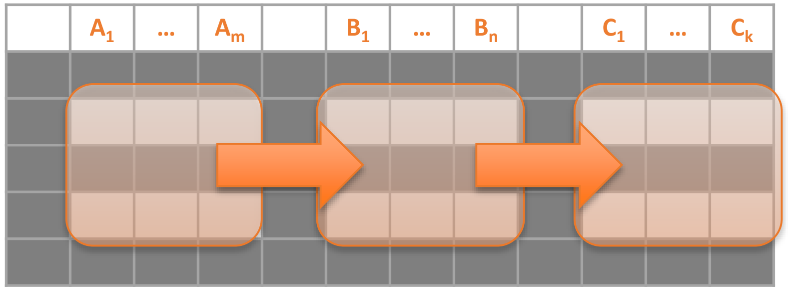


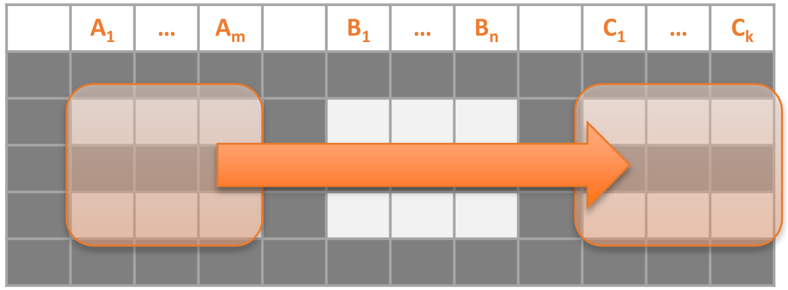


<u>If</u> $A_1, \ldots, A_m \rightarrow B_1,\ldots,B_n$

<u>Then</u> $A_1, \ldots, A_m, C_1, \ldots, C_k \rightarrow B_1,\ldots,B_n, C_1, \ldots, C_k$

Transitivity

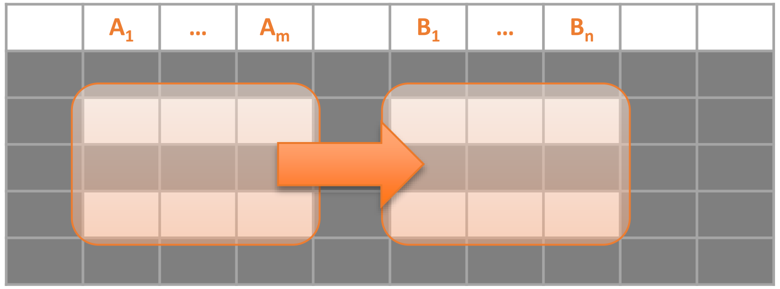


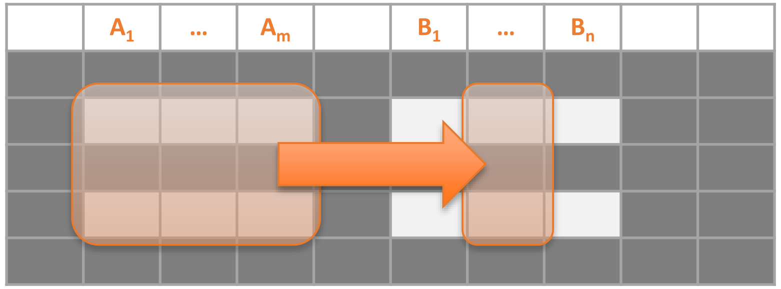


If $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ **and** $B_1, \ldots, B_n \rightarrow C_1, \ldots, C_k$

Then $A_1, \ldots, A_m \rightarrow C_1, \ldots, C_k$

Splitting/Combining Rule





$A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ **<u>is equivalent to *n* FDs</u>** $A_1, \ldots, A_m \rightarrow B_i$ for $i = 1, \ldots, n$

split/combine on the right side, not the left side

Finding Implicit Functional Dependencies

Example:

Products

Name	Color	Category	Dept	Price
Gizmo	Green	Gadget	Toys	49
Widget	Black	Gadget	Toys	59
Gizmo	Green	Whatsit	Garden	99

Provided FD's:

1. {Name} → {Color}
2. {Category} → {Department}
3. {Color, Category} → {Price}

What other FD's hold?
What is a key for this table?

Finding Implicit Functional Dependencies

Inferred FD's	Rules used
4. {Name, Category} -> {Name}	?
5. {Name, Category} -> {Color}	?
6. {Name, Category} -> {Category}	?
7. {Name, Category} -> {Color, Category}	?
8. {Name, Category} -> {Price}	?
9. {Name, Category} -> {Department}	?

Provided FD's:

1. {Name} $\rightarrow$ {Color}
2. {Category} $\rightarrow$ {Department}
3. {Color, Category} $\rightarrow$ {Price}

Finding Implicit Functional Dependencies

Inferred FD's	Rules used
4. {Name, Category} -> {Name}	Trivial
5. {Name, Category} -> {Color}	Transitive (4 -> 1)
6. {Name, Category} -> {Category}	Trivial
7. {Name, Category} -> {Color, Category}	Combine (5 + 6)
8. {Name, Category} -> {Price}	Transitive (7 -> 3)
9. {Name, Category} -> {Department}	Transitive (6 -> 2)

Provided FD's:

1. {Name} $\longrightarrow$ {Color}
2. {Category} $\longrightarrow$ {Department}
3. {Color, Category} $\longrightarrow$ {Price}

Another way to do this: Find the closure of {Name, Category}

Closure of a set of attributes

Given a set of attributes **A_1, …, A_n** and a set of FD's **F:**
the **closure**, **{A_1, …, A_n}$^+$** is the set of attributes **B** s.t. **{A_1, …, A_n}** $\rightarrow$ **B** follows from **F**

Example: **F =**

{name} $\rightarrow$ {color}
{category} $\rightarrow$ {department}
{color, category} $\rightarrow$ {price}

Example Closures:

{name, category}$^+$ = {name, category, color, dept, price}
{name}$^+$ = {name, color}
{category}$^+$ = {category, department}

{name, category} is a superkey

No proper subset of {name, category} is a superkey

{name, category} is a key

To check if A $\longrightarrow$ B, we only need to check if B $\subseteq$ A^+

If A^+ = set of all attributes, then A is a **superkey**

Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$

Repeat until X does not change:

if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F for $\{B_1, \ldots, B_n\} \subseteq X$

then add C to X

Return X as $\{A_1, \ldots, A_n\}^+$

n trivial dependencies

repeatedly apply transitive rule

Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$

Repeat until X does not change:

if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F for $\{B_1, \ldots, B_n\} \subseteq X$

then add C to X

Return X as $\{A_1, \ldots, A_n\}^+$

{name, category}$^+$ =
{name, category}

F =

{name} → {color}
{category} → {dept}
{color, category} → {price}

Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$

Repeat until X does not change:

if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F for $\{B_1, \ldots, B_n\} \subseteq X$

then add C to X

Return X as $\{A_1, \ldots, A_n\}^+$

$\{name, category\}^+ =$
$\{name, category\}$

$\{name, category\}^+ =$
$\{name, category, color\}$

F =

$\{name\} \rightarrow \{color\}$
$\{category\} \rightarrow \{dept\}$
$\{color, category\} \rightarrow \{price\}$

Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$

Repeat until X does not change:

if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F for $\{B_1, \ldots, B_n\} \subseteq X$

then add C to X

Return X as $\{A_1, \ldots, A_n\}^+$

F =

{name} → {color}
{category} → {dept}
{color, category} → {price}

$\{name, category\}^+ =$
$\{name, category\}$

$\{name, category\}^+ =$
$\{name, category, color\}$

$\{name, category\}^+ =$
$\{name, category, color, dept\}$

Closure Algorithm

Start with X = $\{A_1, \ldots, A_n\}$

Repeat until X does not change:

    if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F for $\{B_1, \ldots, B_n\} \subseteq X$

        then add C to X

Return X as $\{A_1, \ldots, A_n\}^+$

F =

{name} → {color}
{category} → {dept}
{color, category} → {price}

$\{\text{name, category}\}^+ =$
{name, category}

$\{\text{name, category}\}^+ =$
{name, category, color}

$\{\text{name, category}\}^+ =$
{name, category, color, dept}

$\{\text{name, category}\}^+ =$
{name, category, color, dept, price}