Functional Dependencies and Normalization

DataBases

Slides from CS145 Stanford (2016), Christopher Ré

Motivation

Design Theory

- Design theory is about how to represent your data to avoid **anomalies**

- Simple algorithms for “best practices”

Designing the Schema of a Database

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

1. Data Anomalies and Constraints

Constraints Prevent (some) Anomalies in the Data

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 <u>redundant</u> information!

Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes anomalies:

Student	Course	Room
Mary	CS145	B01
Joe	CS145	C12
Sam	CS145	B01
..	..	..

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

Constraints Prevent (some) Anomalies in the Data

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

Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes anomalies:

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

...	CS229	C12

Similarly, we can't reserve a room without students = an <u>insert anomaly</u>

Constraints Prevent (some) Anomalies in the Data

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?

Today: develop theory to understand why this design may be better and how to find this decomposition…

2. Functional Dependencies

Functional Dependency

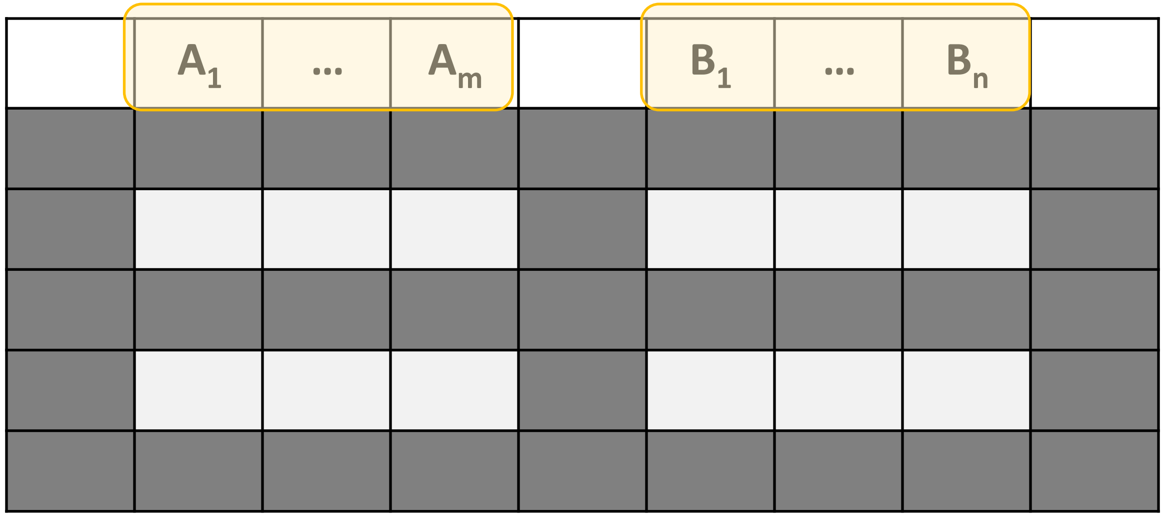
Def: Let A,B be sets of attributes
We write A → B or say A functionally determines B if, for any tuples t_1 and t_2:

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

and we call A → B a <u>functional dependency</u>

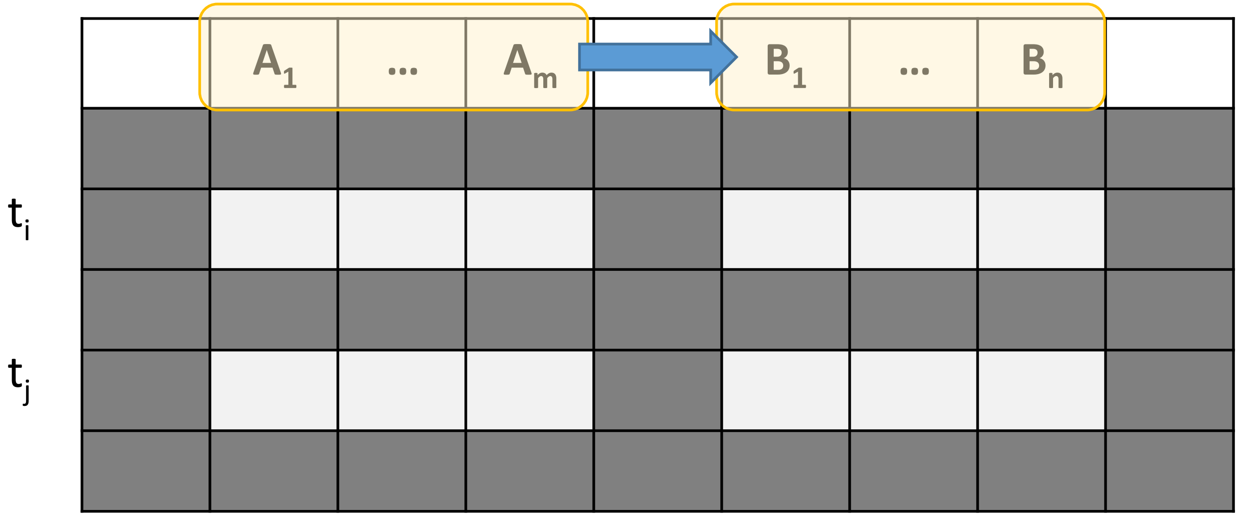
A->B means that
“whenever two tuples agree on A then they agree on B.”

A Picture of FDs



<u>Defn (again):</u>
Given attribute sets $A=\{A_1,\ldots,A_m\}$ and $B = \{B_1,\ldots B_n\}$ in R,

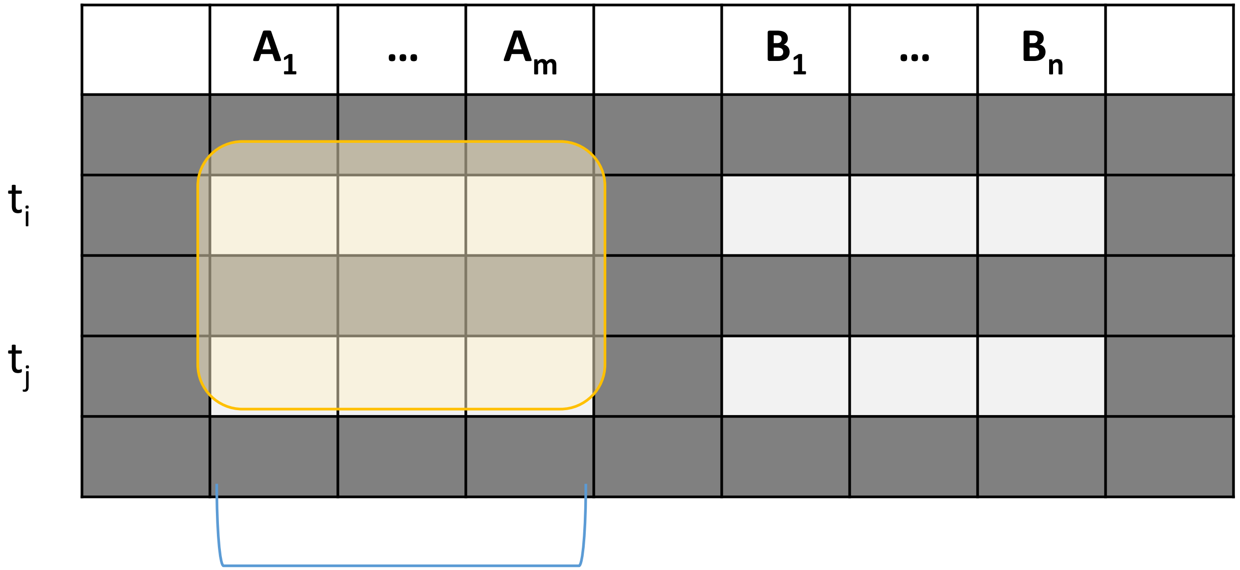
A Picture of FDs



<u>Defn (again):</u>
Given attribute sets $A=\{A_1,\ldots,A_m\}$ and $B = \{B_1,\ldots B_n\}$ in R,

The functional dependency $A\rightarrow B$ on R holds if for any t_i,t_j in R:

A Picture of FDs



If t1,t2 agree here..

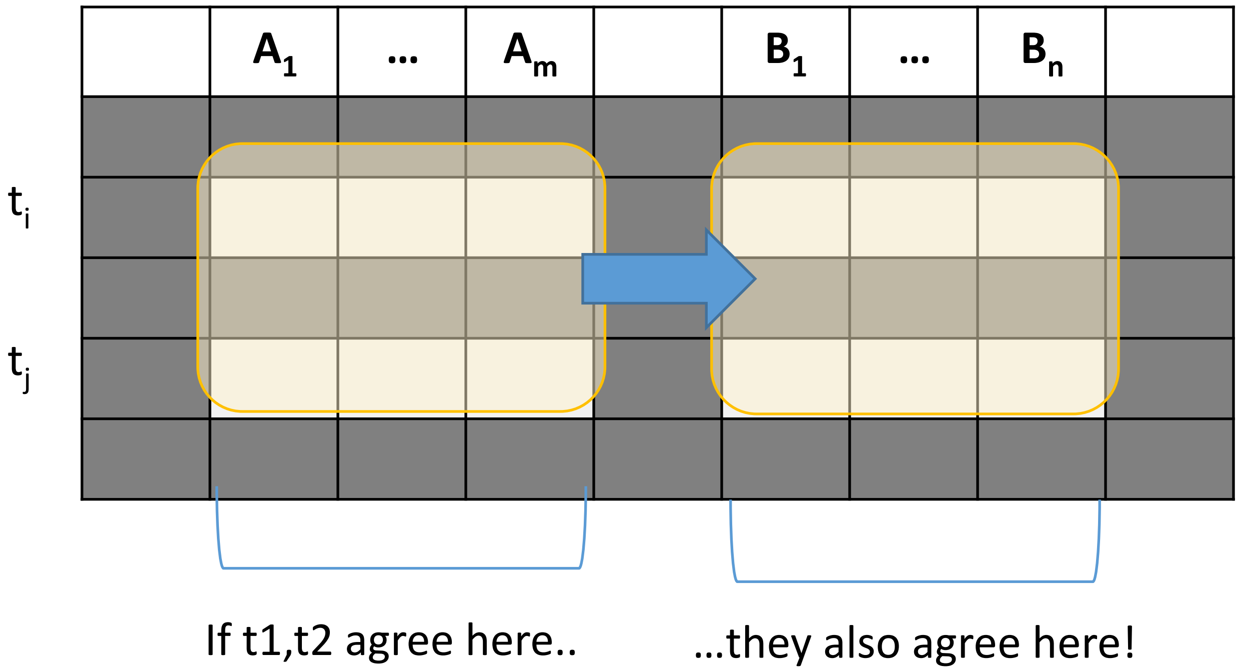
<u>Defn (again):</u>

Given attribute sets $A=\{A_1,\ldots,A_m\}$ and $B = \{B_1,\ldots B_n\}$ in R,

The functional dependency $A \rightarrow B$ on R holds if for any t_i,t_j in R:

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

A Picture of FDs



<u>Defn (again):</u>
Given attribute sets $A=\{A_1,\ldots,A_m\}$ and $B = \{B_1,\ldots B_n\}$ in R,

The functional dependency $A \rightarrow B$ 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 $t_i[A_m] = t_j[A_m]$

<u>then</u> $t_i[B_1] = t_j[B_1]$ AND $t_i[B_2]=t_j[B_2]$ AND … AND $t_i[B_n] = t_j[B_n]$

FDs for Relational Schema Design

- High-level idea: **why do we care about FDs?**

  1. Start with some relational *schema*

  2. Model its *functional dependencies (FDs)*

  3. Use these to *design a better schema*
     1. One which minimizes the possibility of anomalies

Functional Dependencies as Constraints

A **functional dependency** is a form of **constraint**

- *Holds* on some instances not others.
- Part of the schema, helps define a valid *instance*.

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

Recall: an <u>instance</u> of a schema is a multiset of tuples conforming to that schema, i.e. a table

Note: The FD {Course} -> {Room} holds on this instance

Functional Dependencies as Constraints

Note that:

- You can check if an FD is **violated** by examining a single instance;
- However, you **cannot prove** that an FD is part of the schema by examining a single instance.
  - *This would require checking every valid instance*

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

However, cannot prove that the FD {Course} -> {Room} is part of the schema

More Examples

An FD is a constraint which holds, or does not hold on an instance:

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

More Examples

EmpID	Name	Phone		Position
E0045	Smith	1234		Clerk
E3542	Mike	9876	←	Salesrep
E1111	Smith	9876	←	Salesrep
E9999	Mary	1234		Lawyer

{Position} → {Phone}

More Examples

EmpID	Name	Phone	Position
E0045	Smith	1234 →	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234 →	Lawyer

but *not* {Phone} → {Position}

Activity

A	B	C	D	E
1	2	4	3	6
3	2	5	1	8
1	4	4	5	7
1	2	4	3	6
3	2	5	1	8

Find at least three FDs which are violated on this instance:

{ } → { }
{ } → { }
{ } → { }

2.1. Finding Functional Dependencies

“Good” vs. “Bad” FDs

We can start to develop a notion of **good** vs. **bad** FDs:

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

<u>Intuitively:</u>

EmpID -> Name, Phone, Position is “good FD”

- Minimal redundancy, less possibility of anomalies

“Good” vs. “Bad” FDs

We can start to develop a notion of **good** vs. **bad** FDs:

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

Intuitively:

EmpID -> Name, Phone, Position is “good FD”

But Position -> Phone is a “bad FD”

- Redundancy! Possibility of data anomalies

“Good” vs. “Bad” FDs

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

Returning to our original example… can you see how the “bad FD” {Course} -> {Room} could lead to an:

- Update Anomaly
- Insert Anomaly
- Delete Anomaly
- …

Given a set of FDs (from user) our goal is to:

1. Find all FDs, and
2. Eliminate the “Bad Ones".

FDs for Relational Schema Design

- High-level idea: **why do we care about FDs?**

  1. Start with some relational *schema*

  2. Find out its *functional dependencies (FDs)*

  3. Use these to *design a better schema*
     1. One which minimizes possibility of anomalies

This part can be tricky!

Finding Functional Dependencies

- There can be a very **large number** of FDs…
  - *How to find them all efficiently?*

- We can't necessarily show that any FD will hold **on all instances…**
  - *How to do this?*

We will start with this problem:
Given a set of FDs, F, what other FDs must hold?

Finding Functional Dependencies

Equivalent to asking:

Given a set of FDs, $F = \{f_1, \ldots f_n\}$, does an FD g hold?

Inference problem: How do we decide?

Finding Functional Dependencies

<u>Example:</u>

Products

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

Provided FDs:

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

Given the provided FDs, we can see that {Name, Category} → {Price} must also hold on **any instance**…

Which / how many other FDs do?!?

Finding Functional Dependencies

Equivalent to asking:

Given a set of FDs, $F = \{f_1,\ldots f_n\}$, does an FD g hold?

Inference problem: How do we decide?

- Trivial FD
- Non-trivial FD
- Completely non-trivial FD

Finding Functional Dependencies

Equivalent to asking:

Given a set of FDs, $F = \{f_1, \ldots f_n\}$, does an FD g hold?

Inference problem: How do we decide?

Answer: Three simple rules called Armstrong's Rules.

1. Reflexivity: Y is included in X => X -> Y
2. Augmentation: X -> Y => XZ -> YZ
3. Transitivity: X -> Y and Y -> Z => X -> Z

Finding Functional Dependencies

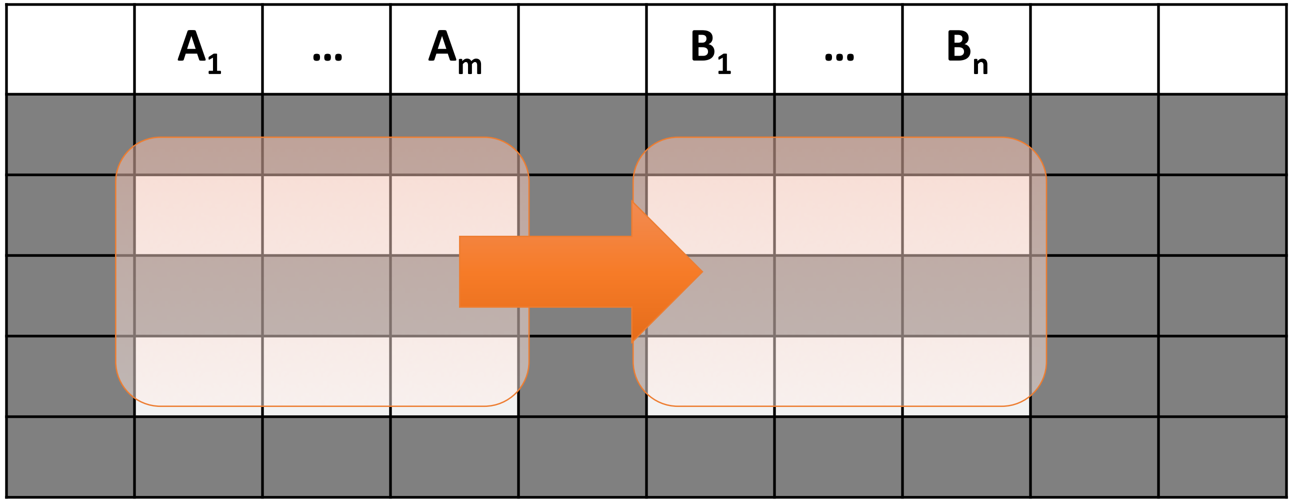
Equivalent to asking:

Given a set of FDs, $F = \{f_1, \ldots f_n\}$, does an FD g hold?

Inference problem: How do we decide?

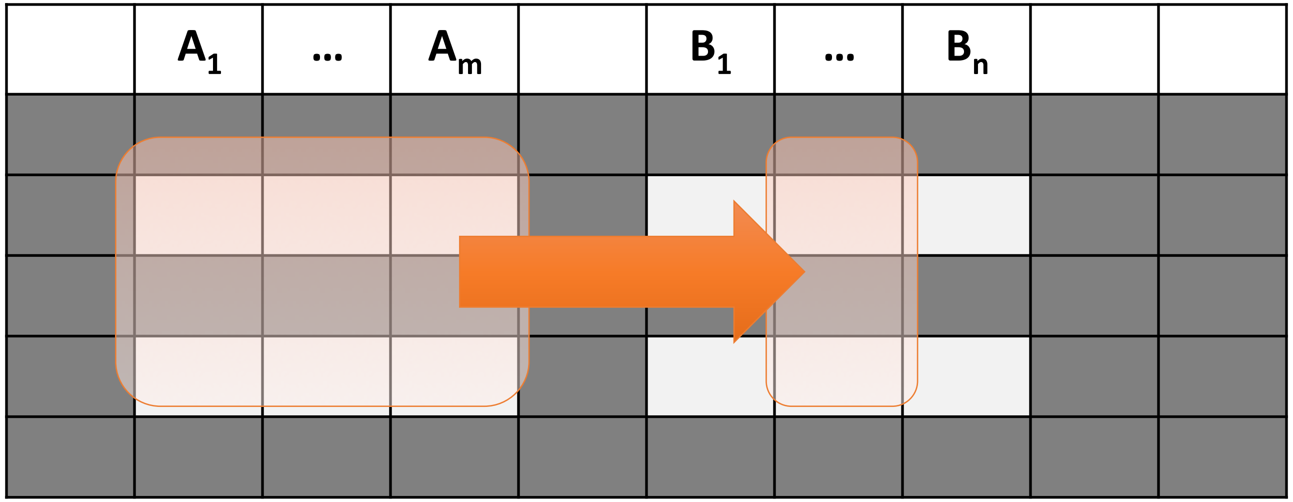
1. Split/Combine
2. Reduction
3. Transitivity

1. Split/Combine



$$A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$$

1. Split/Combine

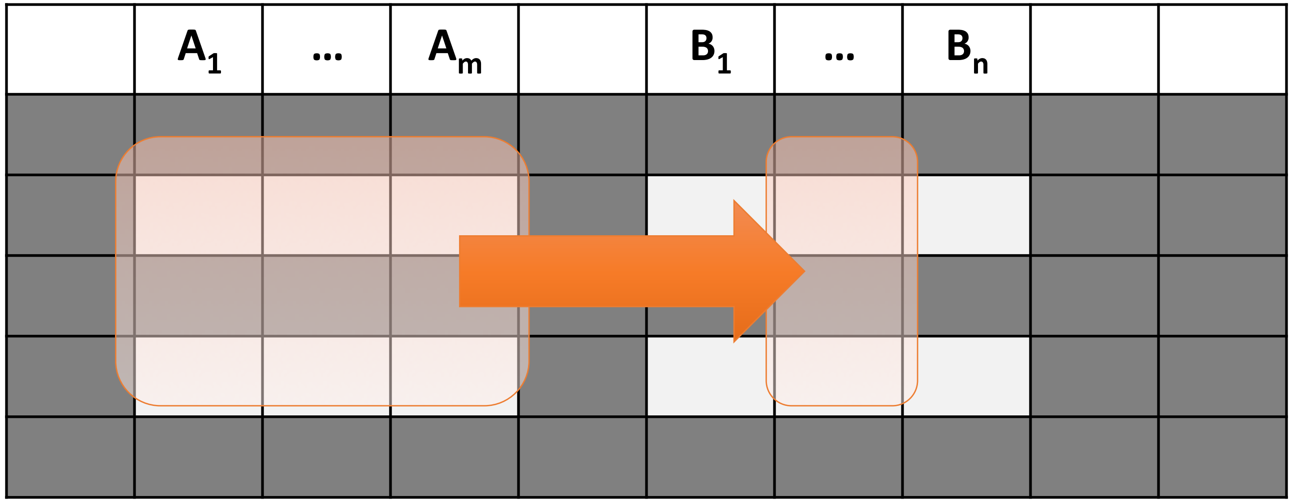


$$A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$$

… is equivalent to the following *n* FDs…

$$A_1, \ldots, A_m \rightarrow B_i \text{ for } i=1,\ldots,n$$

1. Split/Combine

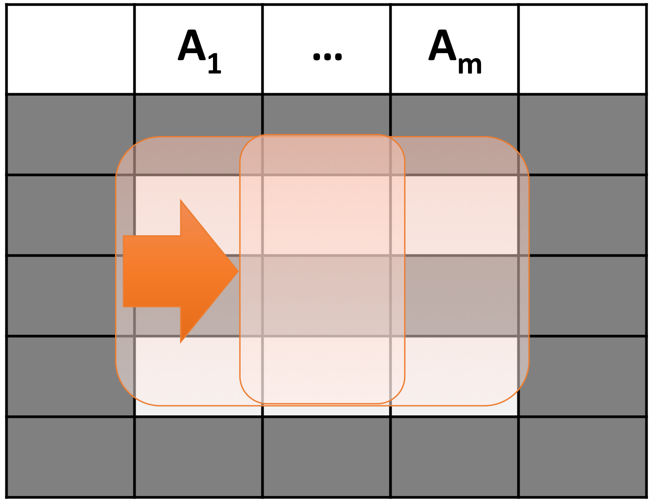


And vice-versa, $A_1,\ldots,A_m \rightarrow B_i$ for $i=1,\ldots,n$

… is equivalent to …

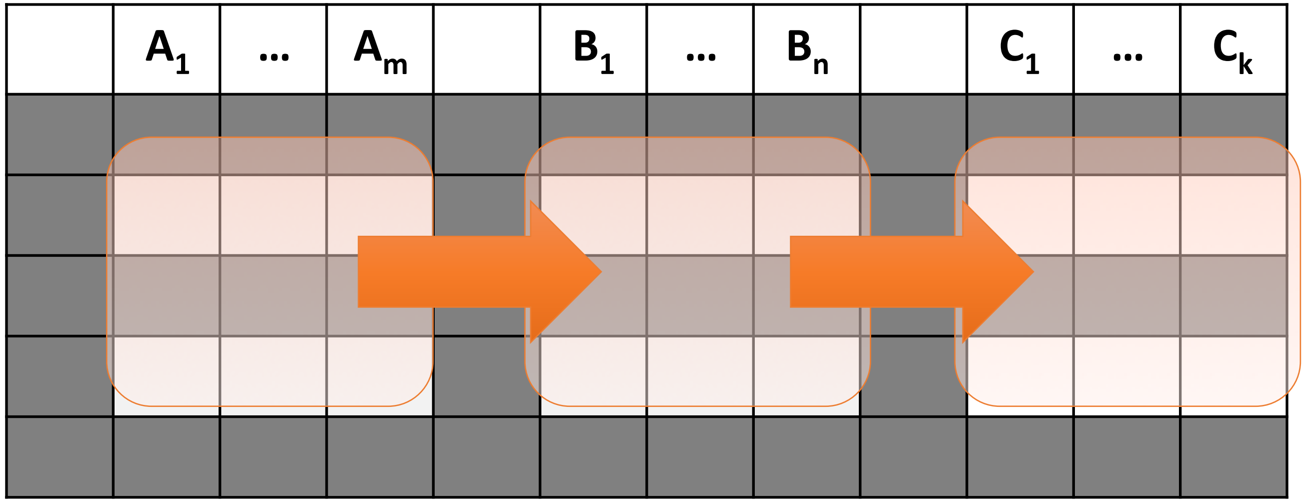
$A_1, \ldots, A_m \rightarrow B_1,\ldots,B_n$

2. Reduction/Trivial



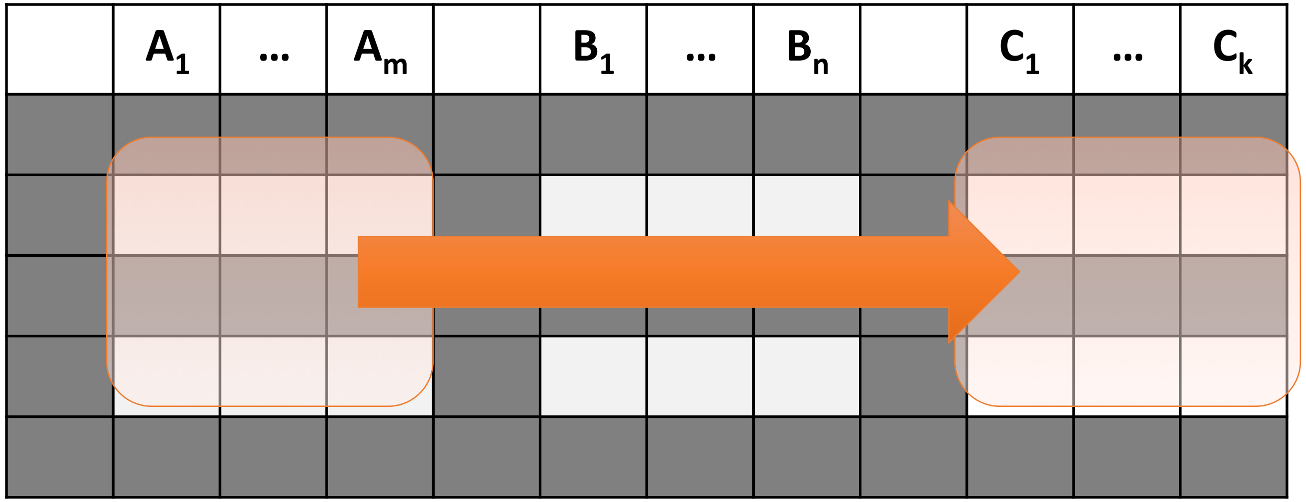
$A_1,\ldots,A_m \rightarrow A_j$ for any $j=1,\ldots,m$

3. Transitivity



$$A_1,\ldots,A_m \rightarrow B_1,\ldots,B_n \text{ and}$$
$$B_1,\ldots,B_n \rightarrow C_1,\ldots,C_k$$

3. Transitivity



$A_1,\ldots,A_m \rightarrow B_1,\ldots,B_n$ and

$B_1,\ldots,B_n \rightarrow C_1,\ldots,C_k$

implies

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

Finding Functional Dependencies

<u>Example:</u>

Products

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

Provided FDs:

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

Which / how many other FDs hold?

Finding Functional Dependencies

Example:

Inferred FDs:

Inferred FD	Rule used
4. {Name, Category} -> {Name}	?
5. {Name, Category} -> {Color}	?
6. {Name, Category} -> {Category}	?
7. {Name, Category -> {Color, Category}	?
8. {Name, Category} -> {Price}	?

Provided FDs:

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

Which / how many other FDs hold?

Finding Functional Dependencies

<u>Example:</u>

Inferred FDs:

Inferred FD	Rule used
4. {Name, Category} -> {Name}	Trivial
5. {Name, Category} -> {Color}	Transitive (4,1)
6. {Name, Category} -> {Category}	Trivial
7. {Name, Category -> {Color, Category}	Split/combine (5,6)
8. {Name, Category} -> {Price}	Transitive (7,3)

Provided FDs:

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

Can we find an algorithmic way to do this?

2.2. Closures

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs F:
Then the <u>closure</u>, $\{A_1, \ldots, A_n\}^+$ is the set of attributes B
s.t. $\{A_1, \ldots, A_n\} \rightarrow B$

<u>Example:</u> F =

```
{name} → {color}
{category} → {department}
{color, category} → {price}
```

Example Closures:

```
{name}+ = {name, color}
{name, category}+ =
{name, category, color, dept, price}
{color}+ = {color}
```

Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$ and set of FDs F.

Repeat until X doesn't change; **do**:

if $\{B_1, \ldots, B_n\} \rightarrow C$ is entailed by F

and $\{B_1, \ldots, B_n\} \subseteq X$

then add C to X.

Return X as X^+

Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change; **do**:
if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$:
then add C to X.
Return X as X^+

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

F =

`{name}` → `{color}`

`{category}` → `{dept}`

`{color, category}` → `{price}`

Closure Algorithm

Start with X = $\{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change; **do**:
if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$:
then add C to X.
Return X as X^+

{name, category}⁺ =
{name, category}

{name, category}⁺ =
{name, category, color}

F =

{name} → {color}

{category} → {dept}

{color, category} → {price}

Closure Algorithm

Start with X = {A_1, …, A_n}, FDs F.
Repeat until X doesn't change; **do**:
if {B_1, …, B_n} → C is in F **and** {B_1, …, B_n} ⊆ X:
then add C to X.
Return X as X^+

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 $X = \{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change; **do**:
  if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$:
    then add C to X.
Return X as X^+

F =

`{name} → {color}`

`{category} → {dept}`

`{color, category} → {price}`

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

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

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

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

Example

`R(A,B,C,D,E,F)`

```
{A,B} → {C}
{A,D} → {E}
{B} → {D}
{A,F} → {B}
```

Compute $\{A,B\}^+ = \{A, B, \qquad\qquad\qquad\qquad \}$

Compute $\{A, F\}^+ = \{A, F, \qquad\qquad\qquad\qquad \}$

Example

```
R(A,B,C,D,E,F)
```

```
{A,B} → {C}
{A,D} → {E}
{B} → {D}
{A,F} → {B}
```

Compute $\{A,B\}^+ = \{A, B, C, D \qquad \}$

Compute $\{A, F\}^+ = \{A, F, B \qquad \}$

Example

```
R(A,B,C,D,E,F)
```

```
{A,B} → {C}
{A,D} → {E}
{B} → {D}
{A,F} → {B}
```

Compute $\{A,B\}^+ = \{A, B, C, D, E\}$

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

3. Closures, Superkeys and Keys

Why Do We Need the Closure?

- With closure we can find all FD's easily

- To check if $X \to A$

  1. Compute X^+

  2. Check if $A \in X^+$

Note here that X is a set of attributes, but A is a single attribute. Why does considering FDs of this form suffice?

Recall the Split/combine rule:
$X \to A_1, \ldots, X \to A_n$
implies
$X \to \{A_1, \ldots, A_n\}$

Using Closure to Infer All FDs

Step 1: Compute X^+, for every set of attributes X:

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

Example:

Given F =

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

No need to compute these- why?

Using Closure to Infer All FDs

Step 1: Compute X^+, for every set of attributes X:

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

Example:

Given F =

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

Step 2: Enumerate all FDs $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \varnothing$:

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

Using Closure to Infer All FDs

Step 1: Compute X^+, for every set of attributes X:

{A}+ = {A}, {B}+ = {B,D}, {C}+ = {C}, {D}+ = {D}, {A,B}+ = {A,B,C,D}, {A,C}+ = {A,C}, {A,D}+ = {A,B,C,D}, {A,B,C}+ = {A,B,D}+ = {A,C,D}+ = {A,B,C,D}, {B,C,D}+ = {B,C,D}, {A,B,C,D}+ = {A,B,C,D}

Example:
Given F =

{A,B} → C
{A,D} → B
{B} → D

Step 2: Enumerate all FDs X → Y, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

{A,B} → {C,D}, {A,D} → {B,C},
{A,B,C} → {D}, {A,B,D} → {C},
{A,C,D} → {B}

"Y is in the closure of X"

Using Closure to Infer All FDs

Step 1: Compute X^+, for every set of attributes X:

{A}+ = {A}, {B}+ = {B,D}, {C}+ = {C}, {D}+ = {D}, {A,B}+ = {A,B,C,D}, {A,C}+ = {A,C}, {A,D}+ = {A,B,C,D}, {A,B,C}+ = {A,B,D}+ = {A,C,D}+ = {A,B,C,D}, {B,C,D}+ = {B,C,D}, {A,B,C,D}+ = {A,B,C,D}

Example:
Given F =

```
{A,B} → C
{A,D} → B
{B}   → D
```

Step 2: Enumerate all FDs X → Y, s.t. Y ⊆ X^+ and $X \cap Y = \varnothing$:

```
{A,B} → {C,D}, {A,D} → {B,C},
{A,B,C} → {D}, {A,B,D} → {C},
{A,C,D} → {B}
```

The FD X → Y is non-trivial

Minimal Cover of a set F of FD

- Minimal subset of elementary FD allowing to generate all the others.

- Theorem:
  - Any set of FD has a minimal cover, that in general is not unique.
- Formally, F is a Minimal Cover iif:
  - All f in F is "elementary".
  - There is no f in F such that F - {f} is "equivalent" to F.

Minimal Cover of a set F of FD

- X -> A is an **elementary** FD if:
  - A is an attribute, X is a set of attributes, A is not included in X
  - it does not exist X' included in X such that X '-> A in F^+
- **Equivalence**
  - Two sets of FD are equivalent if they have the same transitive closure.

3.1. Superkeys and Keys

Keys and Superkeys

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

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

A key is a minimal superkey

Meaning that no subset of a key is also a superkey

Finding Keys and Superkeys

- For each set of attributes X

    1. Compute X^+

    2. If X^+ = set of all attributes then X is a **superkey**

    3. If X is minimal, then it is a **key**

Do we need to check all sets of attributes? Which sets?

Example of Finding Keys

```
Product(name, price, category, color)
```

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

What is a key?

Example of Keys

```
Product(name, price, category, color)
```

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

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

= the set of all attributes

⟹ this is a **superkey**

⟹ this is a **key**, since neither name nor category alone is a superkey

4. Normalization

Normal Forms

- 1st Normal Form (1NF) = All tables are flat
- 2nd Normal Form
- **Boyce-Codd Normal Form (BCNF)**
- **3rd Normal Form (3NF)**

DB designs based on *functional dependencies*, intended to prevent data ***anomalies***

1^{st} Normal Form (1NF)

Student	Courses
Mary	{CS145,CS229}
Joe	{CS145,CS106}
…	…

Violates 1NF.

Student	Courses
Mary	CS145
Mary	CS229
Joe	CS145
Joe	CS106

In 1^{st} NF

1NF Constraint: Types must be atomic!

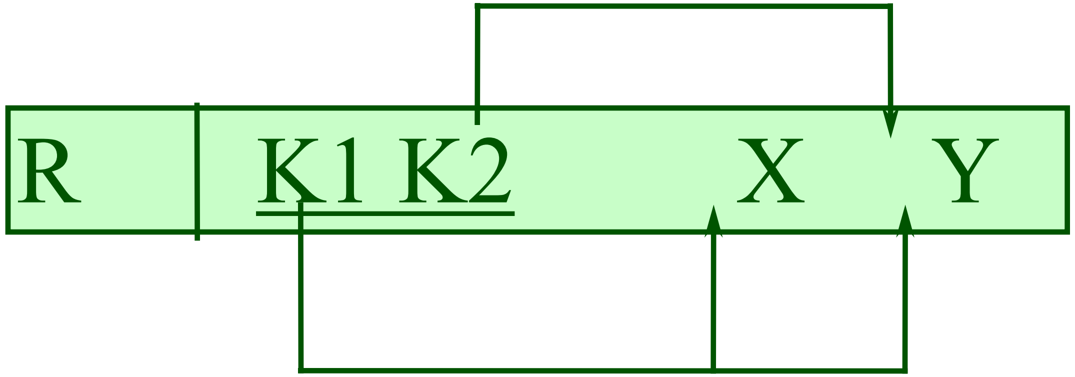
2nd Normal Form (2NF)

Definition

a relationship is in second normal form iff:

- it is the first form
- any non-key attribute is not dependent on a key part

Schema



Such a relationship should be broken into

R1 (<u>K1, K2</u>, X) and R2 (<u>K2</u>, Y)

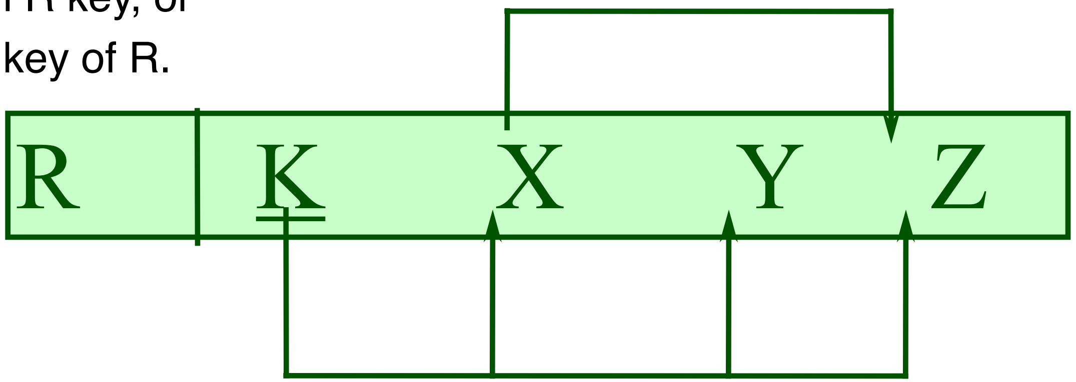
Example 2NF

- Example 1:
  - Supplier (name, address, product, price)
  - The key is (name, product)
  - But name -> address : not second form

- Example 2:
  - R (wine, type, customer, discount)
  - The key is (wine, customer)
  - But wine -> type: not second form

3rd Normal Form (3NF)

- Definition
  - a relationship is in third normal form if for all nontrivial FD in F (X->A) X is a super key or A is a prime attribute (is part of a key).
    - 3NF → 2NF
    - Prohibits FD between non-key attributes (not part of a key)
    - formally:
      - X→A is a nontrivial FD in F and
        - X contains an R key, or
        - A is part of a key of R.
- Diagram



Such a relationship should be broken into

R1 (<u>K</u> X, Y) and R 2 (X, Z)

Example 3NF

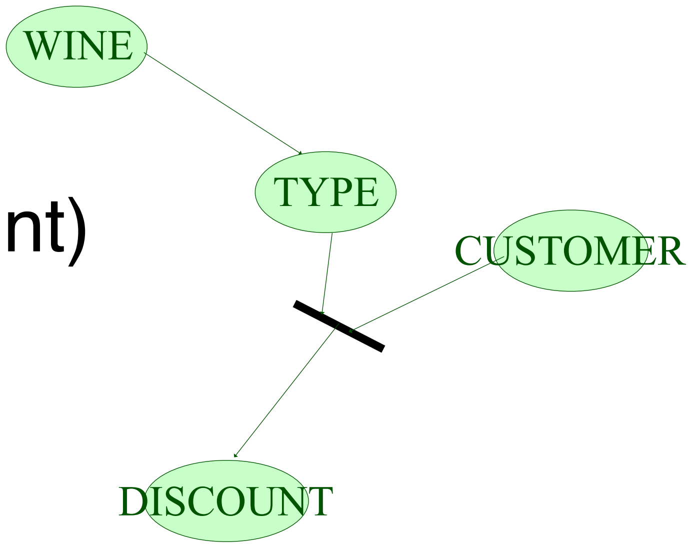
- Example
  - Car (NVH, brand, type, power, color)
  - NVH is key
  - type -> brand
  - type -> Power

  Not in 3rd form!

Decomposition Example

- Car (NVH, brand, type, power, color)
  vehicle (NVH, type, color)
  Model (type, brand, power)

- Reduction (wine, type, customer, discount)
  Discount (type, customer, discount)
  Type (wine, type)
  Order(wine, customer)

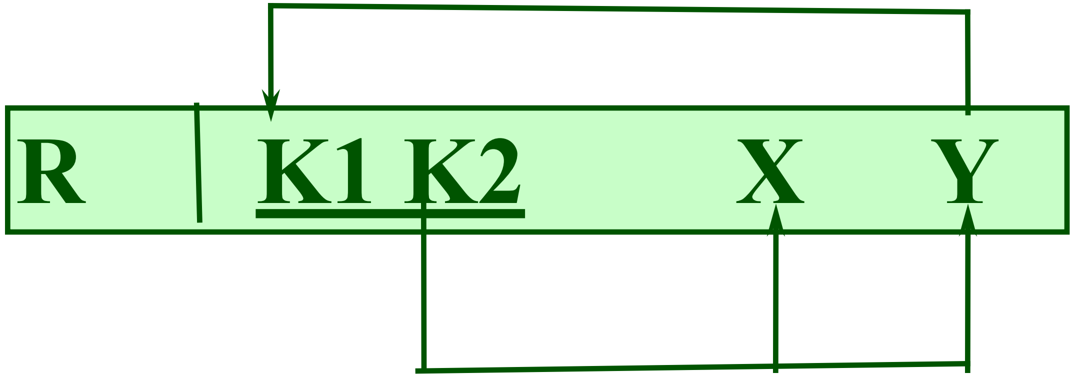


Even Fewer Redundancies: BCNF

Definition

a relationship is in BCNF (Boyce-Codd Normal Form) iff all nontrivial FD in F (X->A) X is a super key

Simpler than 3NF, a little stronger (BCNF -> 3NF)



Such a relationship can be divided into

R1 (<u>K2 Y</u>, X) and R2 (<u>Y</u>, K1)

4.1. Boyce-Codd Normal Form

Back to Conceptual Design

Now that we know how to find FDs, it's a straight-forward process:

1. Search for "bad" FDs
2. If there are any, then *keep decomposing the table into sub-tables* until no more bad FDs
3. When done, the database schema is *normalized*

Recall: there are several normal forms…

Boyce-Codd Normal Form (BCNF)

- Main idea is that we define “good” and “bad” FDs as follows:
  - X → A is a “*good FD” if X is a (super)key*
    - In other words, if A is the set of all attributes
  - X → A is a *“bad FD”* otherwise
- We will try to eliminate the “bad” FDs!

Boyce-Codd Normal Form (BCNF)

- Why does this definition of "good" and "bad" FDs make sense?

- If X is *not* a (super)key, it functionally determines *some* of the attributes

  - Recall: this means there is <u>redundancy</u>
  - And redundancy like this can lead to data anomalies!

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

Boyce-Codd Normal Form

BCNF is a simple condition for removing anomalies from relations:

A relation R is <u>in BCNF</u> if:

if $\{A_1, ..., A_n\} \rightarrow B$ is a non-trivial FD in R

then $\{A_1, ..., A_n\}$ is a superkey for R

Equivalently: $\forall$ sets of attributes X, either ($X^+ = X$) or (X^+ = all attributes)

In other words: there are no "bad" FDs

Example

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

$\{SSN\} \rightarrow \{Name, City\}$

This FD is bad because it is <u>not</u> a superkey

$\Longrightarrow$ **Not** in BCNF

What is the key?
{SSN, PhoneNumber}

Example

Name	SSN	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Madison

SSN	PhoneNumber
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121
987-65-4321	908-555-1234

`{SSN} → {Name,City}`

This FD is now good because it is the key

Let's check anomalies:

- Redundancy ?
- Update ?
- Delete ?

Now in BCNF!

BCNF Decomposition Algorithm

BCNFDecomp(R):

BCNF Decomposition Algorithm

BCNFDecomp(R):
Find a set of attributes X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

Find a set of attributes X which has non-trivial "bad" FDs, i.e. is not a superkey, using closures

BCNF Decomposition Algorithm

BCNFDecomp(R):
  Find a set of attributes X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

  <u>if</u> (not found) <u>then</u> Return R

If no “bad” FDs found, in BCNF!

BCNF Decomposition Algorithm

BCNFDecomp(R):
  Find a set of attributes X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

  if (not found) then Return R

  let $Y = X^+ - X$, $Z = (X^+)^C$

Let Y be the attributes that X functionally determines (+ that are not in X)

And let Z be the other attributes that it doesn't

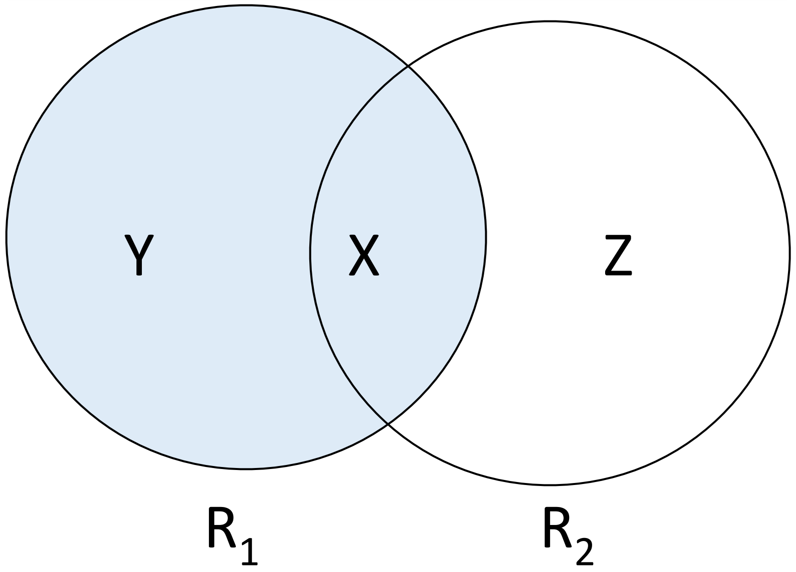
BCNF Decomposition Algorithm

BCNFDecomp(R):
  Find a set of attributes X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

  if (not found) then Return R

  let $Y = X^+ - X$, $Z = (X^+)^C$
  decompose R into $R_1(X \cup Y)$ and $R_2(X \cup Z)$

Split into one relation (table) with X plus the attributes that X determines (Y)…



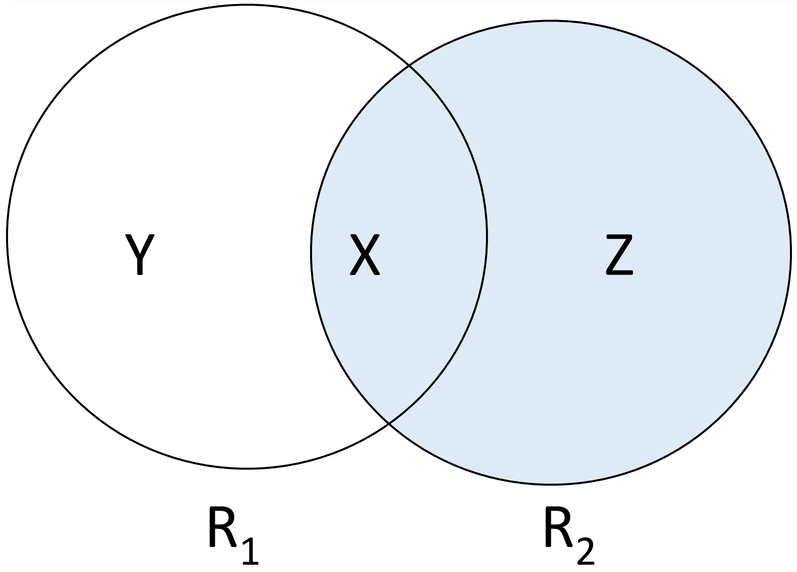
BCNF Decomposition Algorithm

BCNFDecomp(R):
  Find a set of attributes X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

  if (not found) then Return R

  let $Y = X^+ - X$, $Z = (X^+)^C$
  decompose R into $R_1(X \cup Y)$ and $R_2(X \cup Z)$

And one relation with X plus the attributes it does not determine (Z)



BCNF Decomposition Algorithm

BCNFDecomp(R):
   Find a set of attributes X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

   if (not found) then Return R

   let $Y = X^+ - X$, $Z = (X^+)^C$
   decompose R into $R_1(X \cup Y)$ and $R_2(X \cup Z)$

   Return BCNFDecomp(R_1), BCNFDecomp(R_2)

Proceed recursively until no more “bad” FDs!

Example

BCNFDecomp(R):
  Find a set of attributes X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

  if (not found) then Return R

  let $Y = X^+ - X$, $Z = (X^+)^C$
  decompose R into $R_1(X \cup Y)$ and $R_2(X \cup Z)$

  Return BCNFDecomp(R_1), BCNFDecomp(R_2)

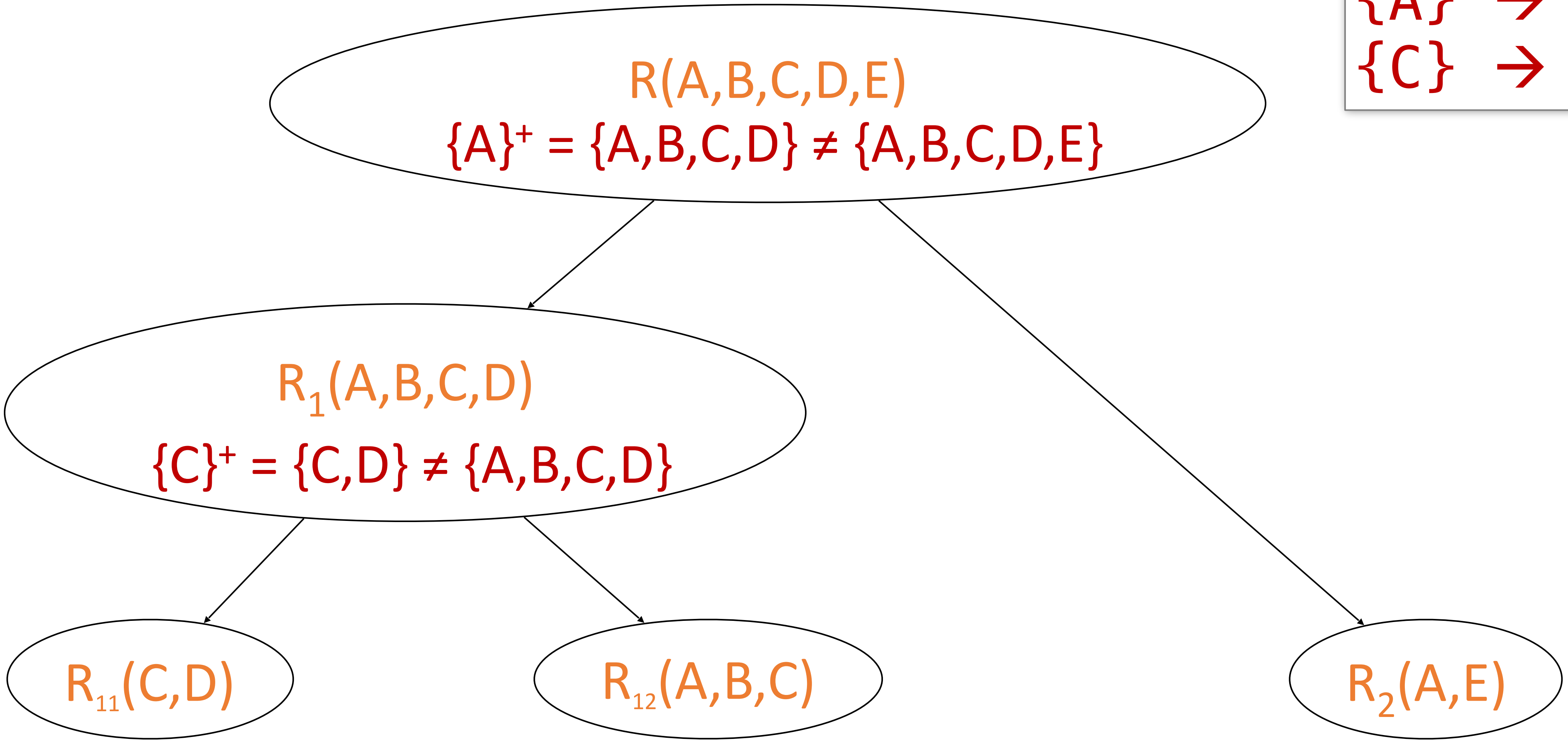
R(A,B,C,D,E)

{A} → {B,C}
{C} → {D}

Example

$R(A,B,C,D,E)$

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



4.2. Decompositions

Recap: Decompose to remove redundancies

1. We saw that **redundancies** in the data (“bad FDs”) can lead to data anomalies

2. We developed mechanisms to **detect and remove redundancies by decomposing tables into BCNF**
    1. BCNF decomposition is *standard practice*- very powerful & widely used!

3. However, sometimes decompositions can lead to **more subtle unwanted effects…**

When does this happen?

Decompositions in General

$$\begin{tikzcd}[column sep=tiny] & \boxed{R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p)} \arrow[dl] \arrow[dr] & \\ \boxed{R_1(A_1,\ldots,A_n,B_1,\ldots,B_m)} & & \boxed{R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)} \end{tikzcd}$$

R_1 = the projection of R on $A_1, \ldots, A_n, B_1, \ldots, B_m$

R_2 = the projection of R on $A_1, \ldots, A_n, C_1, \ldots, C_p$

Theory of Decomposition

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Price
Gizmo	19.99
OneClick	24.99
Gizmo	19.99

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Sometimes a decomposition is "correct"

I.e. it is a <u>Lossless decomposition</u>

Lossy Decomposition

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

However sometimes it isn't

What's wrong here?

Lossless Decompositions

$$R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p)$$

$$R_1(A_1,\ldots,A_n,B_1,\ldots,B_m) \qquad R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)$$

What (set) relationship holds between R1 Join R2 and R if lossless?

Hint: Which tuples of R will be present?

It's lossless if we have equality!

Lossless Decompositions

$$R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p)$$

$$R_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \qquad R_2(A_1, \ldots, A_n, C_1, \ldots, C_p)$$

A decomposition R to (R1, R2) is <u>lossless</u> if R = R1 Join R2

Lossless Decompositions

$R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p)$

$R_1(A_1,\ldots,A_n,B_1,\ldots,B_m)$

$R_2(A_1,\ldots,A_n,C_1,\ldots,C_p)$

If $\{A_1, \ldots, A_n\} \rightarrow \{B_1, \ldots, B_m\}$
Then the decomposition is lossless

Note: don't need $\{A_1, \ldots, A_n\} \rightarrow \{C_1, \ldots, C_p\}$

BCNF decomposition is always lossless. Why?

A Problem with BCNF

Unit	Company	Product
…	…	…

Unit	Company
…	…

Unit	Product
…	…

{Unit} → {Company}

{Unit} → {Company}
{Company,Product} → {Unit}

We do a BCNF decomposition on a "bad" FD:
$\{Unit\}^+ = \{Unit, Company\}$

We lose the FD {Company,Product} → {Unit}!!

So Why is that a Problem?

Unit	Company
Galaga99	UW
Bingo	UW

Unit	Product
Galaga99	Databases
Bingo	Databases

No problem so far. All local FD's are satisfied.

`{Unit} → {Company}`

Unit	Company	Product
Galaga99	UW	Databases
Bingo	UW	Databases

Let's put all the data back into a single table again:

Violates the FD `{Company,Product} → {Unit}`!!

The Problem

- We started with a table R and FDs F

- We decomposed R into BCNF tables R_1, R_2, …
  with their own FDs F_1, F_2, …

- We insert some tuples into each of the relations—which satisfy their local FDs but when reconstruct it violates some FD **across** tables!

<u>Practical Problem</u>: To enforce FD, must reconstruct R—on each insert!

Possible Solutions

- Various ways to handle so that decompositions are all lossless / no FDs lost
  - For example 3NF- stop short of full BCNF decompositions.

- Usually a tradeoff between redundancy / data anomalies and FD preservation…

BCNF still most common- with additional steps to keep track of lost FDs…

5. Other Dependencies

Multi-Value Dependencies (MVDs)

- A multi-value dependency (MVD) is another type of dependency that could hold in our data, **which is not captured by FDs**

Multi-Value Dependencies (MVDs)

- Formal definition:

Given a relation R having attribute sets A, and X, Y s.t. X, Y $\subseteq$ A
The multi-value dependency X->> Y holds on R if
for any tuples t_1, t_2 in R s.t. $t_1[X] = t_2[X]$, there exists a tuple t_3 s.t.:

$t_1[X] = t_2[X] = t_3[X]$

$t_1[Y] = t_3[Y]$

$t_2[A \backslash Y] = t_3[A \backslash Y]$

Example

Movie theater	Film name	Snack
Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
Rains 216	Star Trek: The Wrath of Kahn	Burrito
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
Rains 216	Star Wars: The Boba Fett Prequel	Ramen
Rains 216	Star Wars: The Boba Fett Prequel	Plain pasta

Any FDs?

Example

Movie theater	Film name	Snack
Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
Rains 216	Star Trek: The Wrath of Kahn	Burrito
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
Rains 216	Star Wars: The Boba Fett Prequel	Ramen
Rains 216	Star Wars: The Boba Fett Prequel	Plain pasta

For a given movie theater… given a set of movies and snacks…
Any movie/snack combination is possible!

Example

	Movie theater	Film name	Snack
t_1	Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
t_3	Rains 216	Star Trek: The Wrath of Kahn	Burrito
	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
t_2	Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
	Rains 216	Star Wars: The Boba Fett Prequel	Ramen
	Rains 216	Star Wars: The Boba Fett Prequel	Plain pasta

Example

Movie theater	Film name	Snack
Rains 216	Star Trek: The Wrath of Kahn	Kale Chips
Rains 216	Star Trek: The Wrath of Kahn	Burrito
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Kale Chips
Rains 216	Lord of the Rings: Concatenated & Extended Edition	Burrito
Rains 216	Star Wars: The Boba Fett Prequel	Ramen
Rains 216	Star Wars: The Boba Fett Prequel	Plain pasta

MVD holds over a relation or an instance, so must hold for every applicable pair

Summary

- Constraints allow one to reason about the redundancy in the data

- Normal forms describe how to remove this redundancy by decomposing relations
  - By representing data appropriately, certain errors are essentially impossible