Lecture 7:
Design Theory II

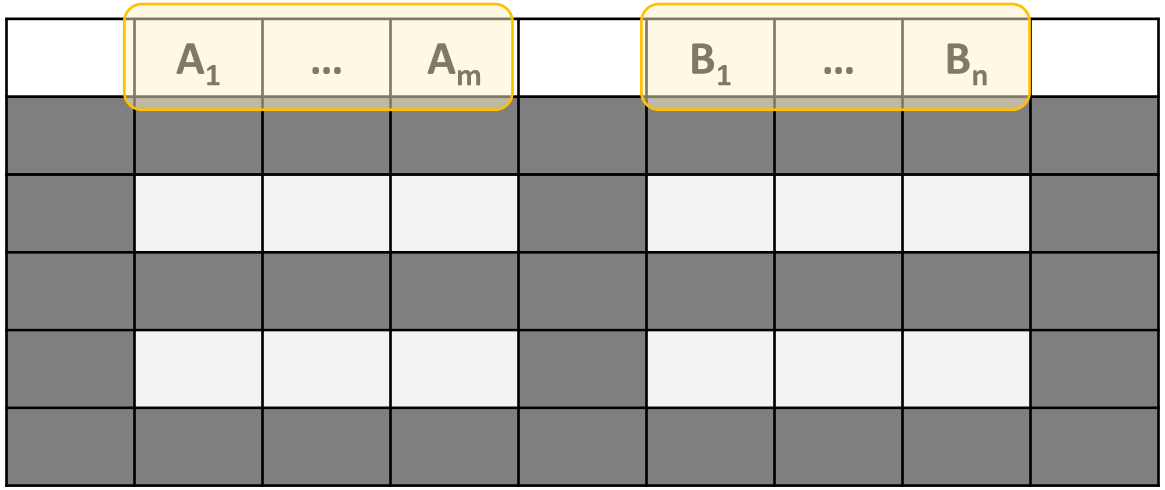
Lecture 7:
Design Theory II

Today's Lecture

1. Recap from Previous Lecture
2. Boyce-Codd Normal Form
   - ACTIVITY
3. Decompositions
   - ACTIVITY

2. Recap from Previous Lecture

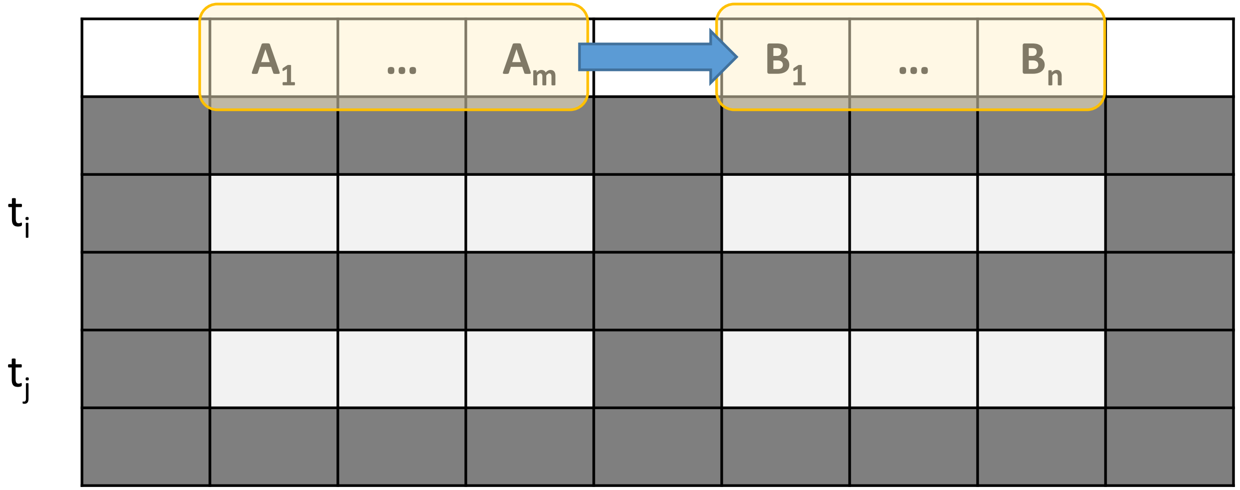
Functional Deps



Defn:

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

Functional Deps

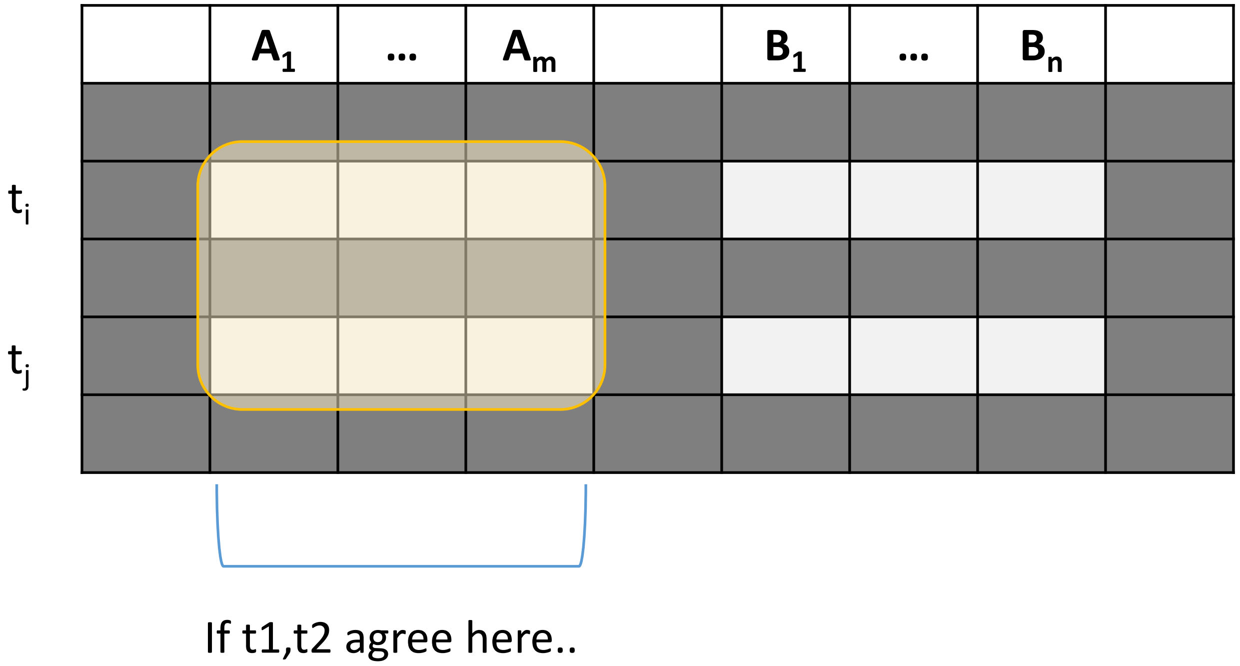


Defn:

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:

Functional Deps



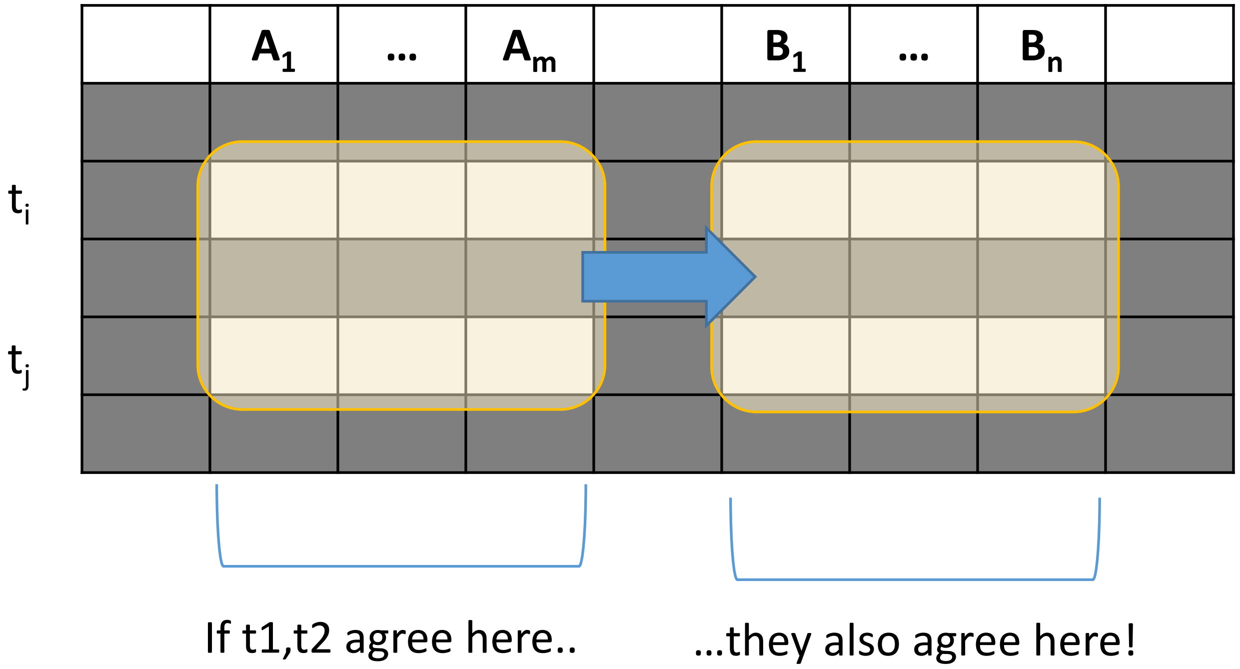
<u>Defn:</u>

Given attribute sets **$A=\{A_1,\dots,A_m\}$** and **$B = \{B_1,\dots B_n\}$** 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 $t_i[A_m] = t_j[A_m]$

Functional Deps



<u>Defn:</u>

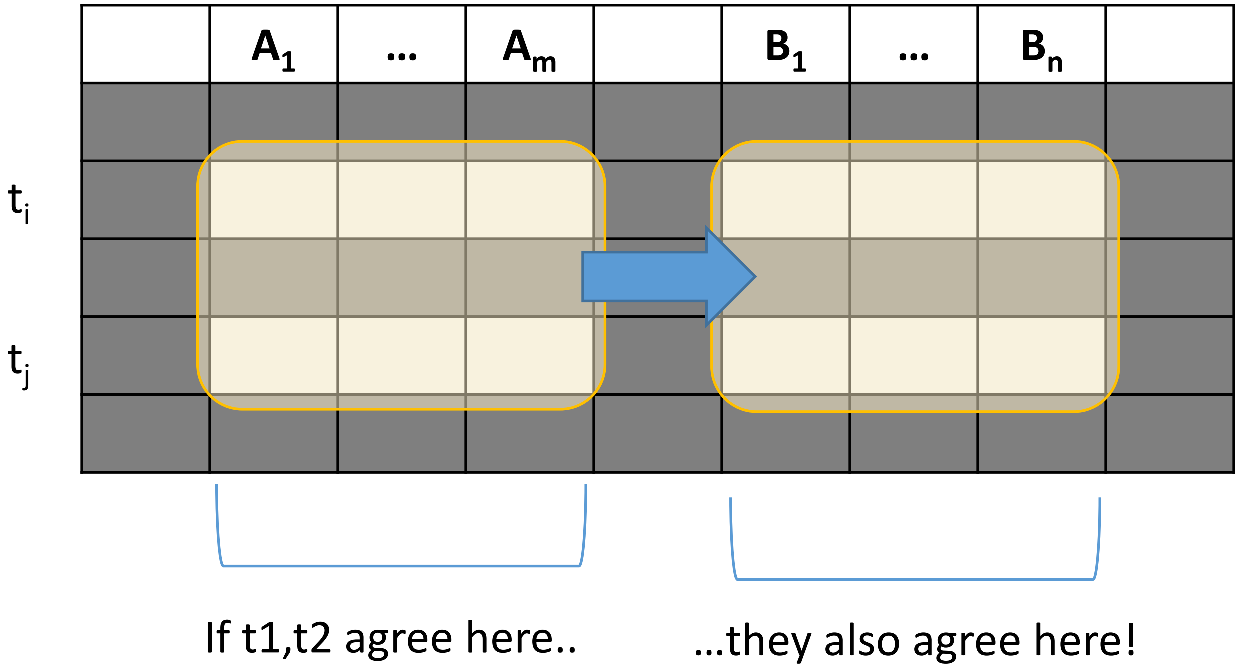
Given attribute sets **A={A_1,…,A_m}** and **B = {B_1,…B_n}** in **R,**

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 $t_i[A_m] = t_j[A_m]$

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

Functional Deps



<u>Defn:</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 $t_i[B_n] = t_j[B_n]$

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. Split/Combine,
2. Reduction, and
3. Transitivity

Closure of a set of Attributes

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

Example: 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^+

Finding Functional Dependencies

1. Use Armstrong's rules to find FDs that hold

Armstrong's Rules.
1. Split/Combine,
2. Reduction, and
3. Transitivity

2. Use Closure Alg to find ALL FDs

Step 0: Give a set of FDs F

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

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

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**

Putting it all together

1. FDs impose constraints on data. They prevent anomalies.

2. They can be used to find the closure of a set of attributes.

3. The Closure algorithm allows us to identify superkeys and keys.

2. Boyce-Codd Normal Form

What you will learn about in this section

1. Conceptual Design

2. Boyce-Codd Normal Form

3. The BCNF Decomposition Algorithm

4. ACTIVITY

Conceptual Design

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 **in BCNF** 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} → {Name,City}`

This FD is *bad* because it is **not** 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

Now in BCNF!

`{SSN} → {Name,City}`

This FD is now *good* because it is the key

Let's check anomalies:

- Redundancy ?
- Update ?
- Delete ?

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]

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

  <u>**let**</u> $Y = X^+ - X$, $Z = (X^+)^C$

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***

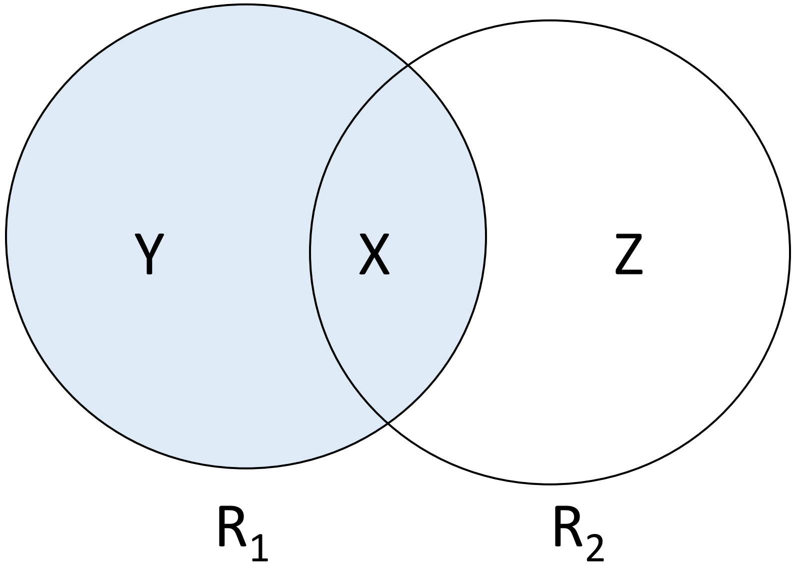
BCNF Decomposition Algorithm

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)…



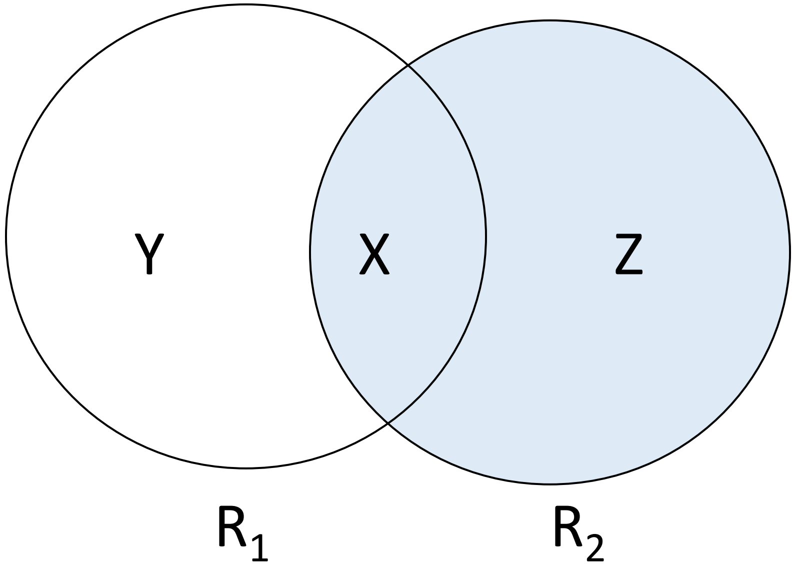
BCNF Decomposition Algorithm

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

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

  <u>**let**</u> $Y = X^+ - X$, $Z = (X^+)^C$
  decompose R into $\mathbf{R_1(X \cup Y)}$ and $\mathbf{R_2(X \cup Z)}$

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]

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

  <u>**let**</u> $Y = X^+ - X$, $Z = (X^+)^C$
  decompose R into $\mathbf{R_1(X \cup Y)}$ and $\mathbf{R_2(X \cup Z)}$

  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]

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

<u>**let**</u> $Y = X^+ - X$, $Z = (X^+)^C$

decompose R into $\mathbf{R_1(X \cup Y)}$ and $\mathbf{R_2(X \cup Z)}$

Return BCNFDecomp(R_1), BCNFDecomp(R_2)

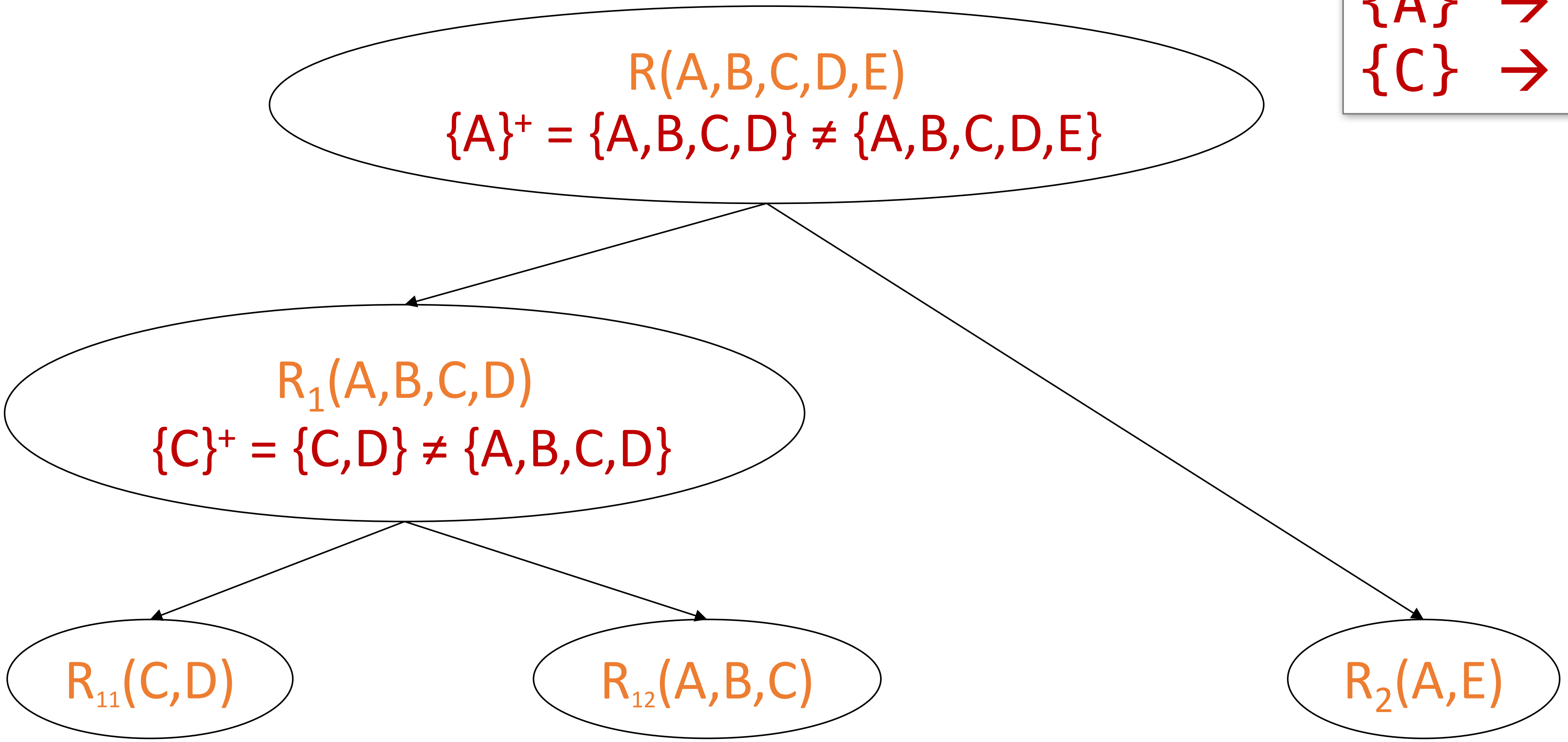
R(A,B,C,D,E)

{A} → {B,C}
{C} → {D}

Example

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

{A} → {B,C}
{C} → {D}



Activity-7.ipynb Exercise 1

3. 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

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

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 **Lossless decomposition**

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

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

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

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

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

$$\begin{array}{ccc} & R(A_1,\ldots,A_n,B_1,\ldots,B_m,C_1,\ldots,C_p) & \\ \swarrow & & \searrow \\ R_1(A_1,\ldots,A_n,B_1,\ldots,B_m) & & R_2(A_1,\ldots,A_n,C_1,\ldots,C_p) \end{array}$$

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

Lossless Decompositions

$R(A_1,...,A_n,B_1,...,B_m,C_1,...,C_p)$

$R_1(A_1,...,A_n,B_1,...,B_m)$

$R_2(A_1,...,A_n,C_1,...,C_p)$

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

Problem: To enforce a FD, must reconstruct original relation—*on each insert!*

Note: This is historically inaccurate, but it makes it easier to explain

A Problem with BCNF

Unit	Company	Product
…	…	…

Unit	Company
…	…

Unit	Product
…	…

$\{Unit\} \rightarrow \{Company\}$

$\{Company, Product\} \rightarrow \{Unit\}$

We do a BCNF decomposition on a "bad" FD:

$\{Unit\}^+ = \{Unit, Company\}$

$\{Unit\} \rightarrow \{Company\}$

We lose the FD $\{Company, Product\} \rightarrow \{Unit\}$!!

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!

Practical Problem: To enforce FD, must reconstruct R—*on each insert!*

Dependency Preserving Decompositions

- Given **R** and a set of FDs *F*, we decompose **R** into **R1** and **R2**. Suppose:
  - **R1** has a set of FDs *F1*
  - **R2** has a set of FDs *F2*
  - *F1* and *F2* are computed from *F*

A decomposition is **<u>dependency preserving</u>** if by enforcing *F1* over **R1** and *F2* over **R2**, we can enforce *F* over **R**

Good example

Person(SSN, name, age, canDrink)

- $SSN \longrightarrow name, age$
- $age \longrightarrow canDrink$

decomposes into

- $\mathbf{R_1}$(SSN, name, age)
  - $SSN \longrightarrow name, age$
- $\mathbf{R_2}$(age, canDrink)
  - $age \longrightarrow canDrink$

Bad example

R(A, B, C)

- $A \longrightarrow B$
- $B, C \longrightarrow A$

Decomposes into:

- **R_1**(A, B)
  - $A \longrightarrow B$
- **R_2**(A, C)
  - no FDs here!!

R_1

A	B
a_1	b
a_2	b

R_2

A	C
a_1	c
a_2	c

recover

A	B	C
a_1	b	c
a_2	b	c

The recovered table violates $B, C \longrightarrow A$

Possible Solutions

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

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

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

Activity-7.ipynb Exercise 2