CS411
Database Systems

04: Relational Schema Design

Kazuhiro Minami

Primary Goal: Minimize Redundancy

- Basic approach: decompose an original schema into sub-schemas
  - $R(A_1,\ldots,A_n) \Rightarrow S(B_1,\ldots,B_m)$ and $T(C_1,\ldots,C_k)$ such that $\{A_1,\ldots,A_n\} = \{B_1,\ldots,B_m\} \cup \{C_1,\ldots,C_k\}$
- Challenges:
  - Avoid information loss
  - Easy to check functional dependencies (FDs)
  - Ensure good query performance

Normal Forms

Define the condition that guarantees the desired properties of a relation schema

- Boyce Codd Normal Form (BCNF)
- Third Normal Form (3NF)
- Fourth Normal Form (4NF)

Others...

Boyce-Codd Normal Form

A relation R is in BCNF if whenever there is a nontrivial FD $A_1 \ldots A_n \rightarrow B$ for R, $\{A_1 \ldots A_n\}$ is a superkey for R.

An FD is *trivial* if all the attributes on its right-hand side are also on its left-hand side.

SSN	Name	Phone Number
123-32-1099	Fred	(201) 555-1234
123-32-1099	Fred	(206) 572-4312
909-43-4444	Joe	(908) 464-0028
909-43-4444	Joe	(212) 555-4000
234-56-7890	Jocelyn	(212) 555-4000

FD: SSN $\rightarrow$ Name

What are the keys?

The only key is {SSN, Phone Number}.

How do I know? Augmentation + minimality.

Is it in BCNF?

No. SSN is not a key.

What about that alternative schema we recommended earlier---are they in BCNF?

<u>SSN</u>	Name
123-32-1099	Fred
909-43-4444	Joe

Important FDS: SSN → Name
Keys: {SSN}.
Is it in BCNF? Yes.

<u>SSN</u>	<u>Phone Number</u>
123-32-1099	(201) 555-1234
123-32-1099	(206) 572-4312
909-43-4444	(908) 464-0028
909-43-4444	(212) 555-4000

If Phone Number → SSN holds

Important FDS:
Phone Number → SSN.
Keys: {Phone Number}
Is it in BCNF? Yes.

If Phone Number → SSN doesn't hold

Important FDS: none.
Keys: {SSN, Phone Number}
Is it in BCNF? Yes.

What about that alternative schema we recommended earlier---are they in BCNF?

<u>SSN</u>	Name
123-32-1099	Fred
909-43-4444	Joe

<u>SSN</u>	<u>Phone Number</u>
123-32-1099	(201) 555-1234
123-32-1099	(206) 572-4312
909-43-4444	(908) 464-0028
909-43-4444	(212) 555-4000

True or False:

Any 2-attribute relation is in BCNF.

Name $\rightarrow$ Price, Category
What are the keys for this one?
Is it in BCNF?

Name	Price	Category
Gizmo	$19.99	gadgets
OneClick	$24.99	toys

A relation R is in BCNF if whenever there is a nontrivial FD A1 ... An $\rightarrow$ B for R, {A1 ... An} is a superkey for R.

Name $\rightarrow$ Price, Category
What are the keys for this one?
Is it in BCNF?

Name	Price	Category
Gizmo	$19.99	gadgets
OneClick	$24.99	toys

Answers: Key = {Name}, it's in BCNF, true.

Just breaking a relation schema into two-attribute subsets could cause information loss

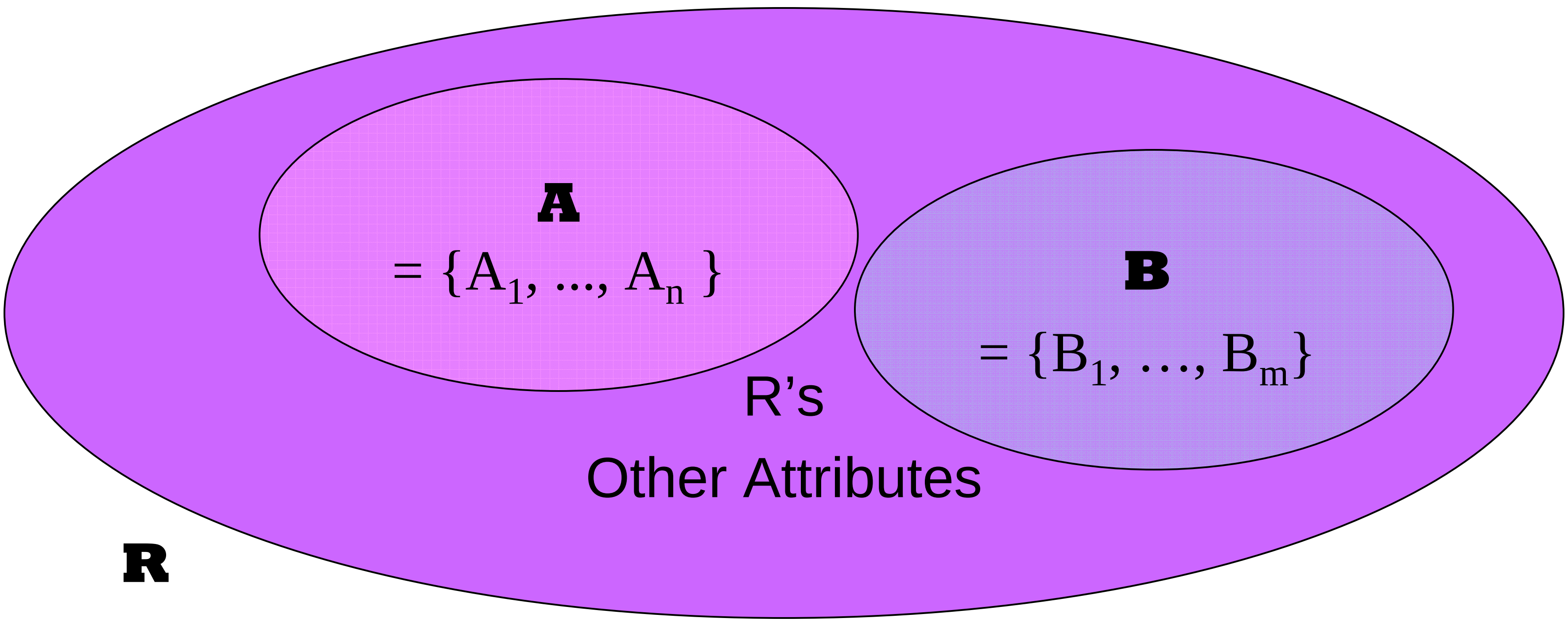
Q: Is this a good idea?

$$R(A_1,\ldots,A_n) \Rightarrow R_1(A_1,A_2),\ \ldots,\ R_{n/2}(A_{n-1},A_n)$$

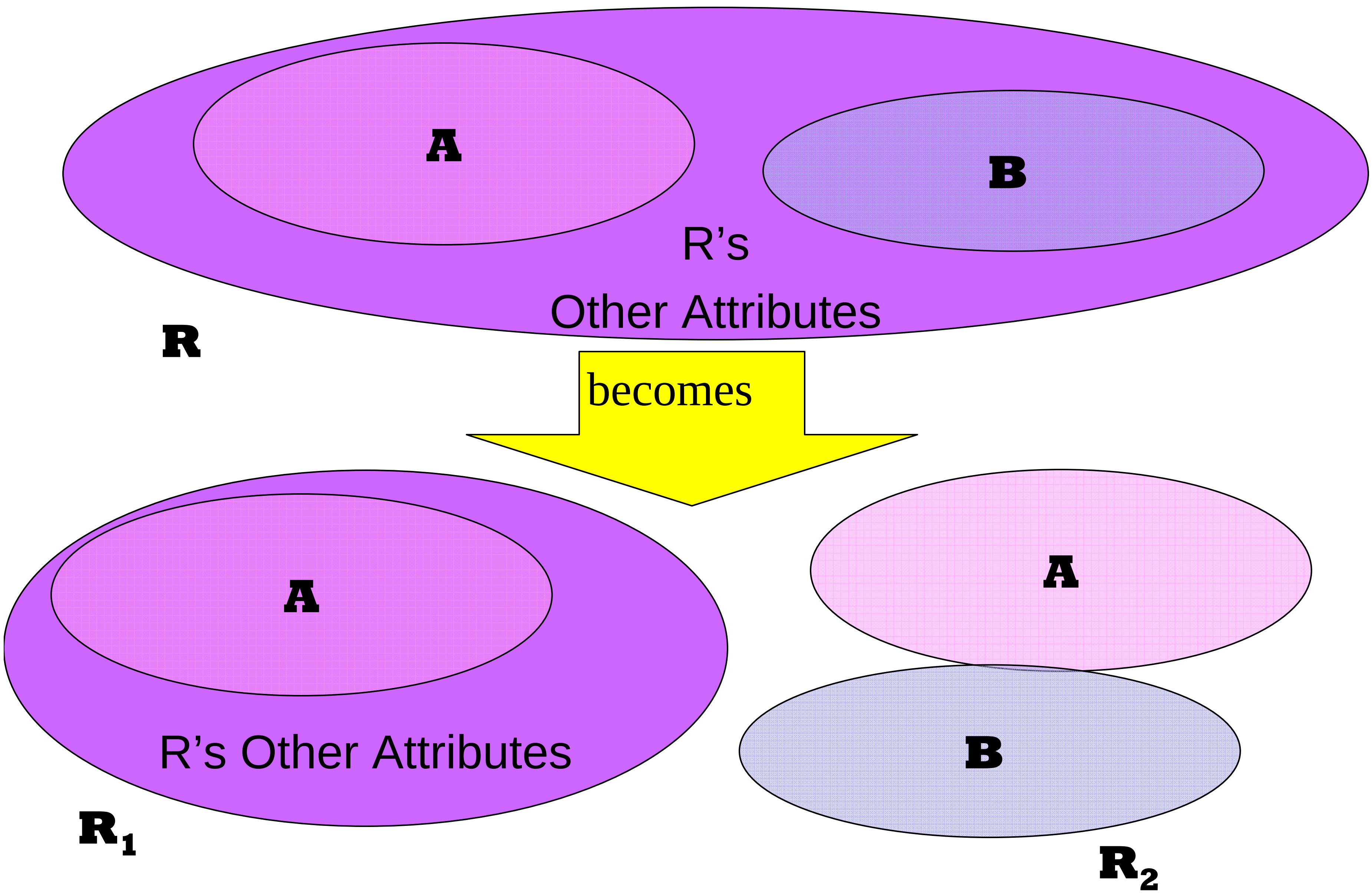
If relation R is not in BCNF, you can pull out the violating part(s) until it is.

1. Find a dependency that violates BCNF:

$$\mathbf{A} \rightarrow \mathbf{B}$$



2. Break R into R1 and R2 as follows.



3. Repeat until all relations are in BCNF.

Heuristic to speed things up and reduce the final number of relations:

Make B as large as possible!

NetID	Name

NetID	Address

NetID	Height

NetID	EyeColor

NetID	HairColor

won’t give as good query performance as

NetID	Name	Address	Height	EyeColor	HairColor

Can you turn this one into BCNF?

PERSON

NetID	Name	Birthdate	EyeColor	Parent	CanVote

Functional dependencies:

NetID → Name, Birthdate, EyeColor, CanVote

Birthdate → CanVote

The key is {NetID, Parent} so this FD violates BCNF

PERSONINFO

NetID	Name	Birthdate	EyeColor	Parent

VOTING

Birthdate	CanVote

But this FD is still violated, so we are not in BCNF yet

One more split needed to reach BCNF

PERSON

NetID	Name	Birthdate	EyeColor	Parent	CanVote

Functional dependencies:

NetID $\rightarrow$ Name, Birthdate, EyeColor, CanVote

Birthdate $\rightarrow$ CanVote

PERSONINFO2

NetID	Name	Birthdate	EyeColor

We split the old PersonInfo into two relations. Now everything is in BCNF.

PARENTINFO

NetID	Parent

VOTING

Birthdate	CanVote

An Official BCNF Decomposition Algorithm

Compute the closures of every subset of attributes in R

Input: relation R, set S of FDs over R.

Output: a set of relations in BCNF.

1. Compute keys for R (from from S).
2. Use S^+ and keys to check if R is in B

Heuristics to reduce the amount of work

   a. Pick a violation FD $A \rightarrow B$.

   b. Expand B as much as possible, by computing A^+.

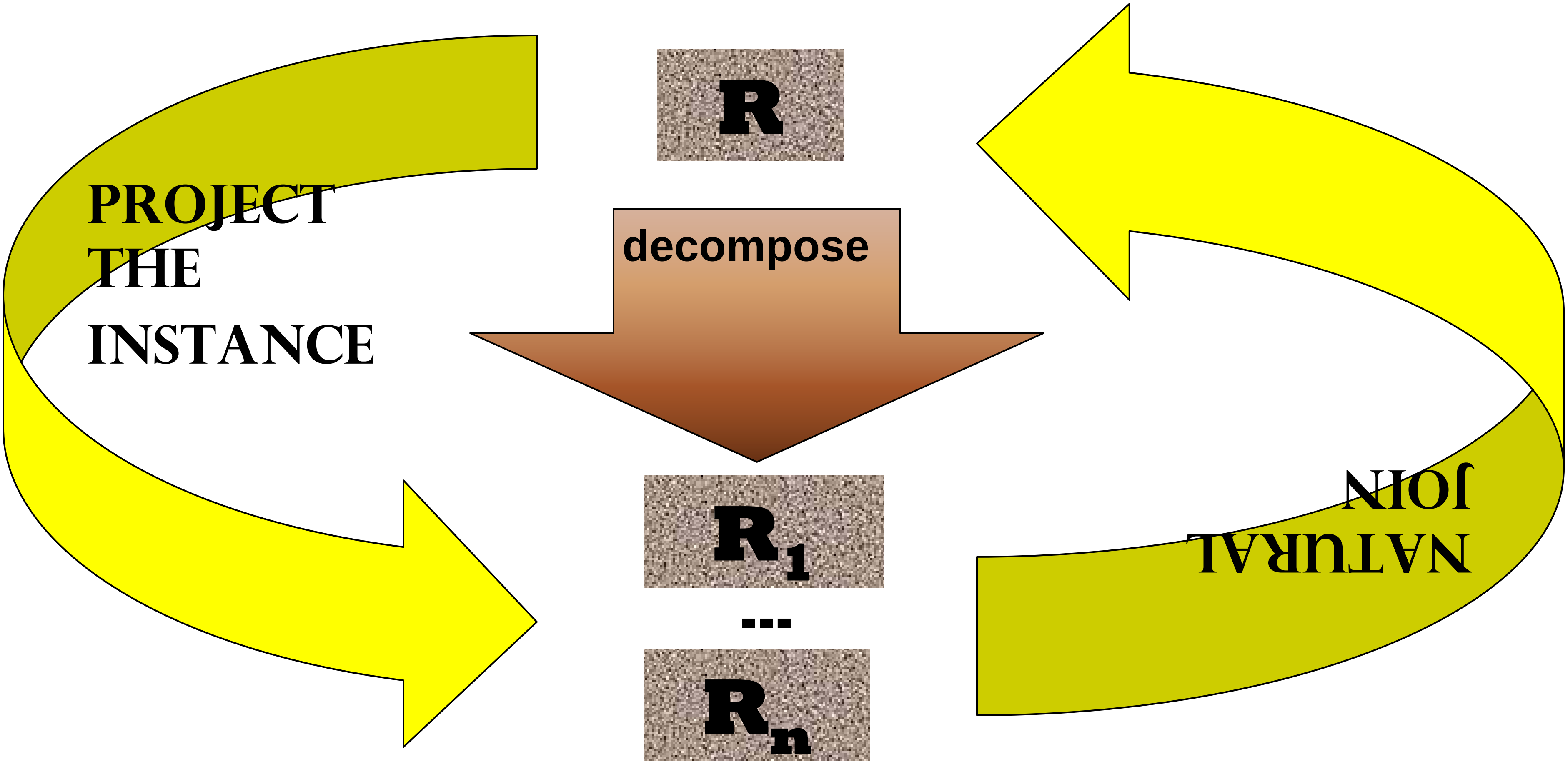
   c. Create $R1 = A^+$, and $R2 = A \cup (R - A^+)$.

   d. Find the FDs over R1, using S^+. Repeat for R2.

   e. Recurse on R1 & its set of FDs. Repeat for R2.

3. Else R is already in BCNF; add R to the output.

Any good schema decomposition should be *lossless*.



Lossless iff a trip around the outer circle gives you back exactly the original instance of R.

Natural Join is the only way to restore the original relation

- R=

A	B
X	Y
X	Z
Y	Z
Z	V

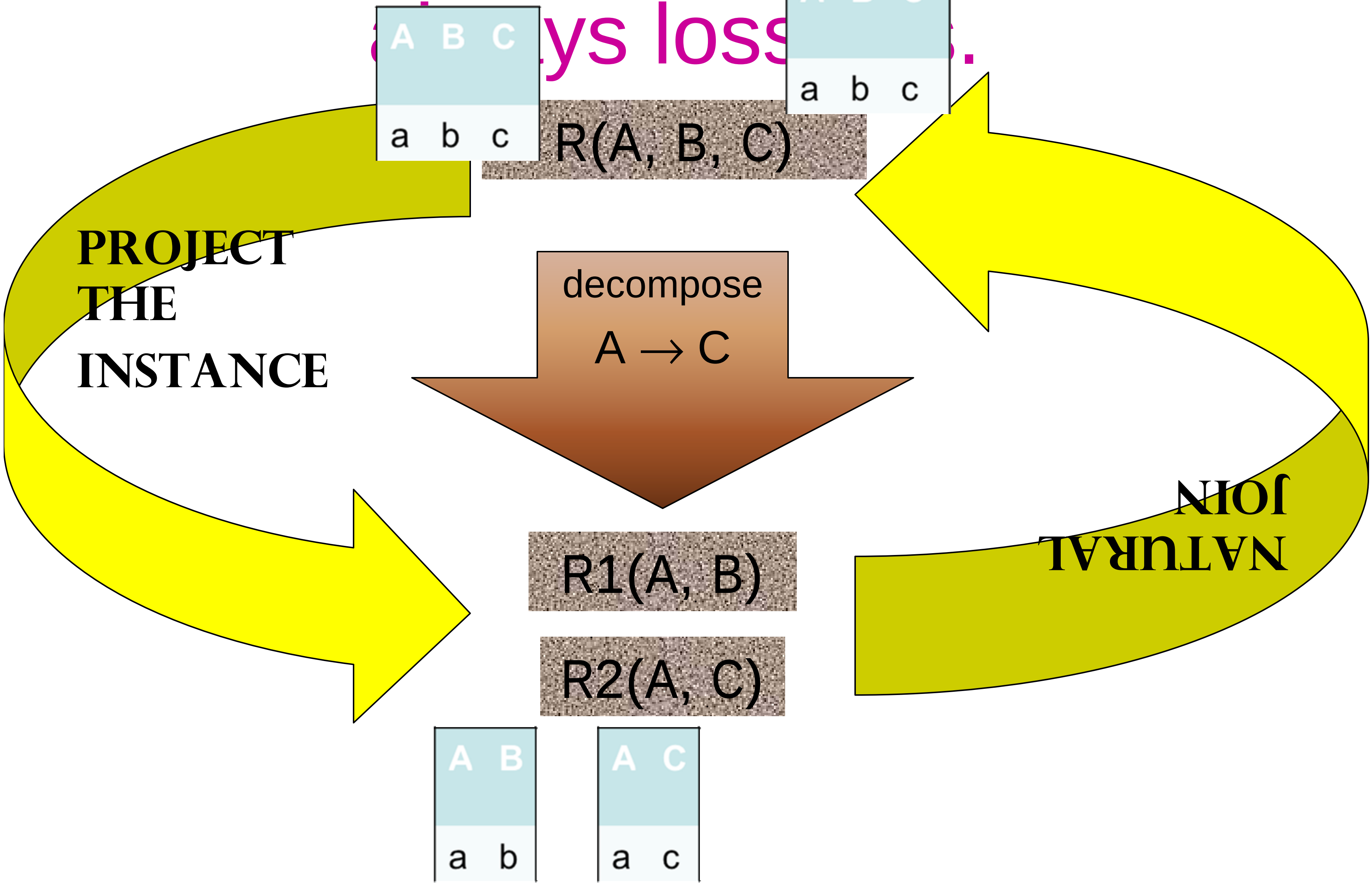
S=

B	C
Z	U
V	W
Z	V

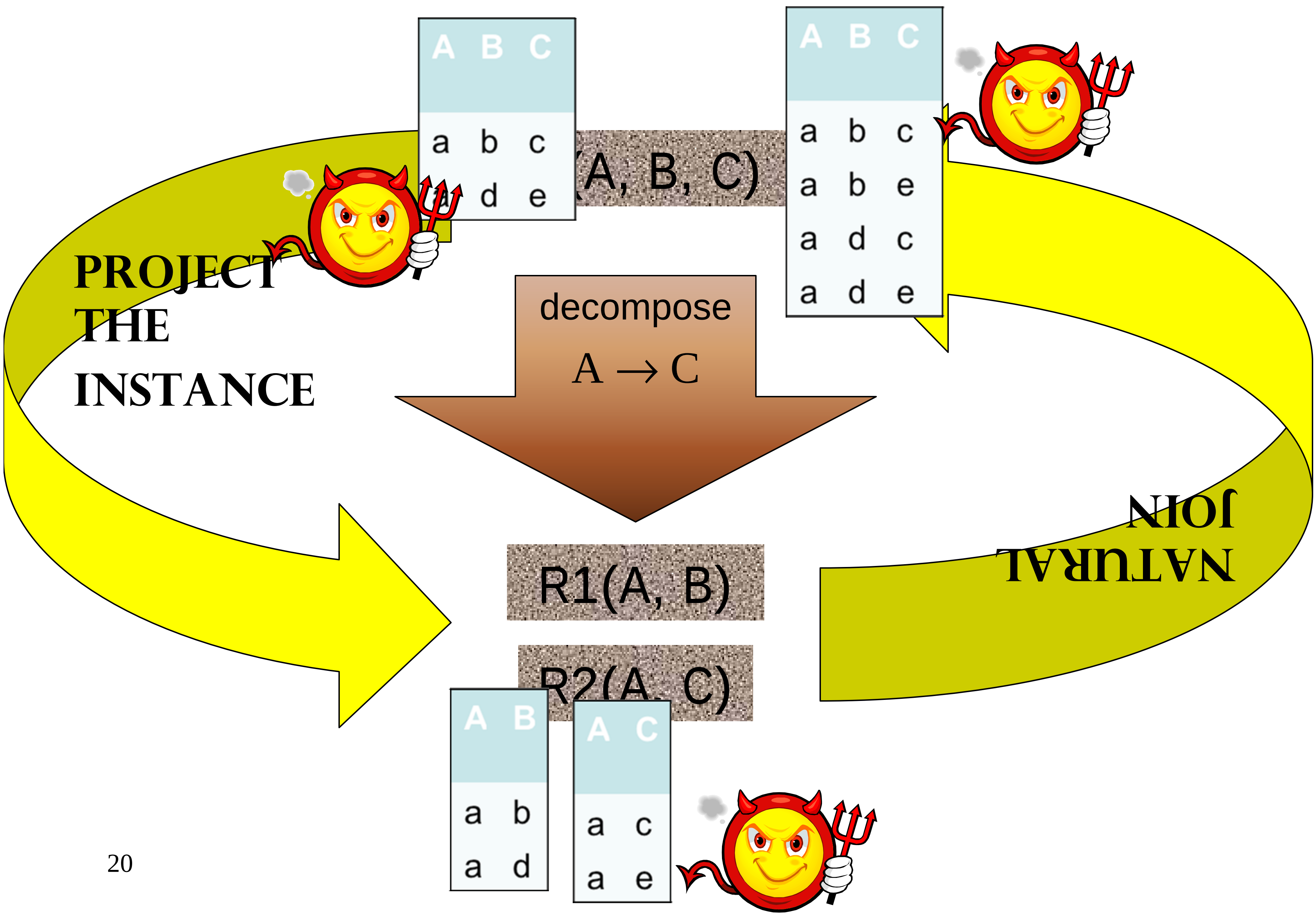
- $R \bowtie S$ =

A	B	C
X	Z	U
X	Z	V
Y	Z	U
Y	Z	V
Z	V	W

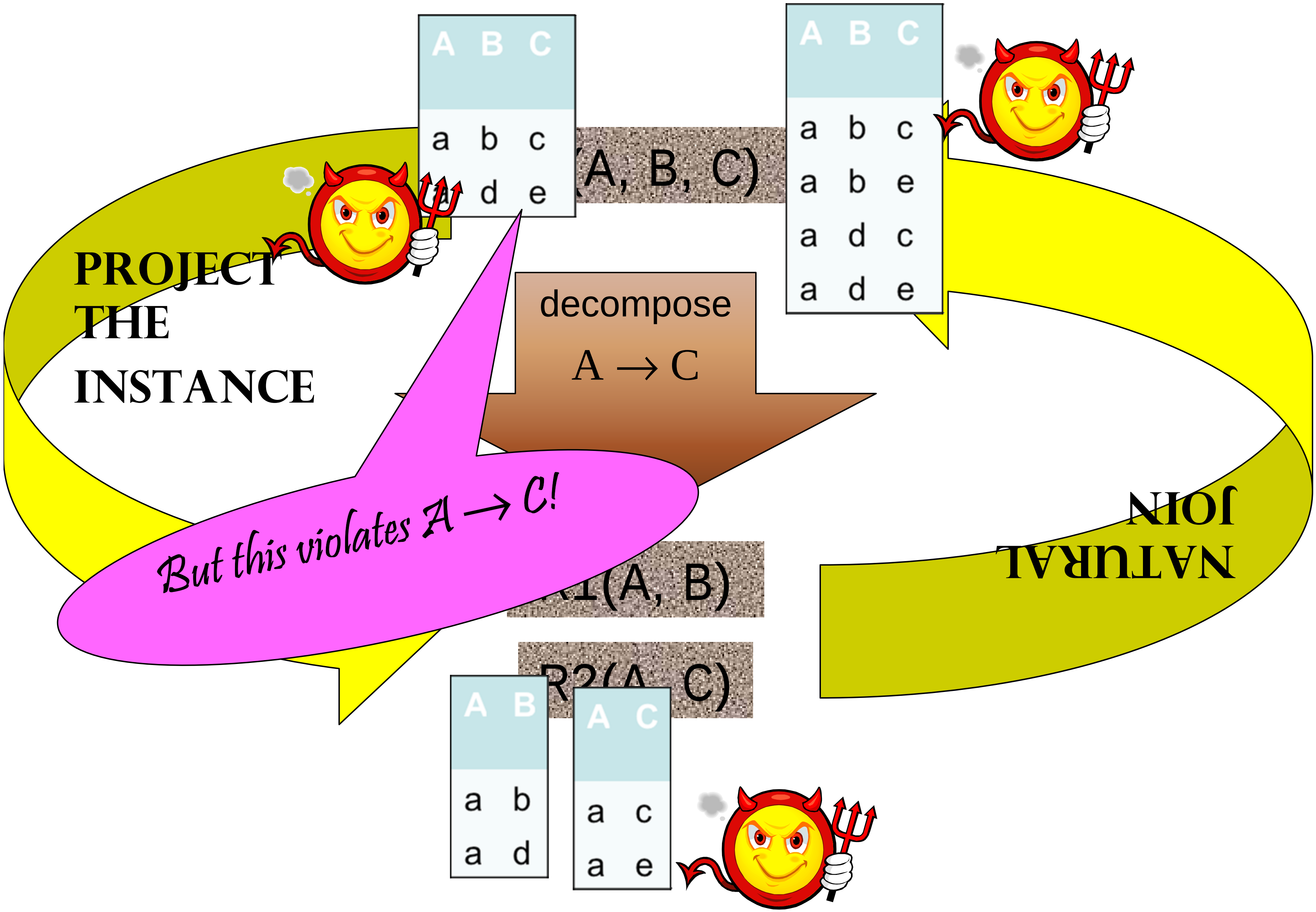
BCNF decompositions are always lossless.



Why don't we get garbage?



Why don’t we get garbage?



BCNF doesn't always have a *dependency-preserving* decomposition.

A schema doesn't *preserve dependencies* if you ***have*** to do a join to check an FD

Account	Client	Office
111	Papa John's	Champaign
334	Papa John's	Madison
121	Papa Del's	Champaign
242	Garcia's	Champaign

Client, Office $\rightarrow$ Account

Account $\rightarrow$ Office

Key is {Client, Office}

violates BCNF

decompose into BCNF

Account	Office
111	Champaign
334	Madison
121	Champaign
242	Champaign

Account $\rightarrow$ Office

Account	Client
111	Papa John's
334	Papa John's
121	Papa Del's
242	Garcia's

No nontrivial FDs

Can't check this FD now without doing a join

A schema *does* preserve dependencies if you can check each FD with decomposed relations

A	B	C

$A \rightarrow B$

$B \rightarrow C$

Key = {A}

violates BCNF

decompose into BCNF

What about $A \rightarrow C$? Do we have to do a join to check it?

No.

A	B

$A \rightarrow B$

B	C

$B \rightarrow C$

So this BCNF decomposition does preserve dependencies.

Normal Forms

First Normal Form = all attributes are atomic

Second Normal Form (2NF) = old and obsolete

Boyce Codd Normal Form (BCNF)

Third Normal Form (3NF)

Fourth Normal Form (4NF)

Others...

If a BCNF decomposition doesn't preserve dependencies, use **3rd Normal Form** instead.

R is in 3NF
if for every nontrivial FD $A_1, \ldots, A_n \rightarrow B$,
either $\{A_1, \ldots, A_n\}$ is a superkey,
or B is part of a key.

Weakens BCNF.

Synthesis Algorithm for 3NF Schemas

1. Find a *minimal basis G* of the set of FDs for relation R
2. For each FD $X \rightarrow A$ in G, add a relation with attributes XA
3. If none of the relation schemas from Step 2 is a superkey for R, add a relation whose schema is a key for R

Result will be lossless and will preserve dependencies.
Result will be in 3NF, but might not be in BCNF.

Minimal Basis

A set of FD's F is a minimal basis of a set of dependencies E if

1. $E = F^+$
2. Every dependen ute for its right-hand sid
3. Cannot remove r remove attribute FD in F (minimality)

We only need to check whether FD's in a minimal basis is preserved in decomposed relations

Example:

$E = \{A \rightarrow B,\ A \rightarrow C,\ B \rightarrow A,\ B \rightarrow C,\ C \rightarrow A,\ C \rightarrow B\}$

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

Normal Forms

First Normal Form = all attributes are atomic

Second Normal Form (2NF) = old and obsolete

Boyce Codd Normal Form (BCNF)

Third Normal Form (3NF)

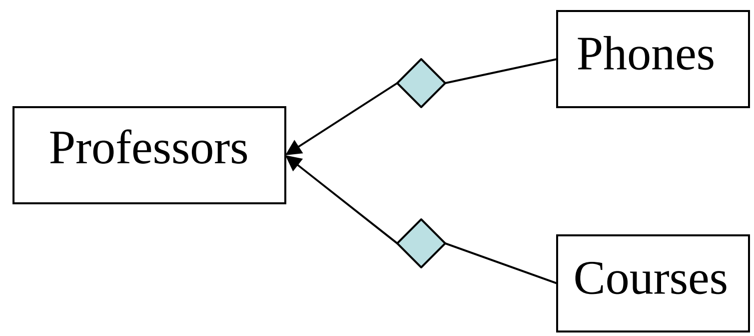
Fourth Normal Form (4NF)

Others...

BCNF doesn't catch every kind of redundancy (much less every bad schema)

Every combination of phone numbers and my courses

NetID	Phone	Course
winslett	333-3333	CS 511
winslett	123-4567	CS 411
winslett	333-3333	CS 411
winslett	123-4567	CS 511



Multivalued dependencies capture this kind of redundancy.

NetID $\twoheadrightarrow$ Phone Number
NetID $\twoheadrightarrow$ Course

Definition of Multi-valued Dependency

A1 … An $\twoheadrightarrow$ B1 … Bm holds iff

	A1	…	An	B1	…	Bm	C1	…	Ck
t	a1	…	an	b11	…	bm1	c11	…	ck1
u	a1	…	an	b12	…	bm2	c12	…	ck2
v	a1	…	an	b11	…	bm1	c12	…	ck2

Whenever two tuples agree on the A's,

there must be a tuple that agrees with them on the A's,

agrees with one of them on the B's,

and agrees with the other one of them on the C's.

You can tear apart a relation R with an MVD.

If A1 … An ↠ B1 … Bm holds in R,
then the decomposition

R1(A1, …, An, B1,…, Bm)

R2(A1, …, An, C1 ,…, Ck)

is lossless.

A1	…	An	B1	…	Bm
a1	…	an	b11	…	bm1
a1	…	an	b12	…	bm2

A1	…	An	C1	…	Ck
a1	…	an	c11	…	ck1
a1	…	an	c12	…	ck2

Note: an MVD A1 … An ↠ B1 … Bm
implicitly talks about "the other" attributes C1, …, Ck.

The inference rules for MVDs are not the same as the ones for FDs.

The most basic one:

If $A1 \ldots An \rightarrow B1 \ldots Bm$,

then $A1 \ldots An \twoheadrightarrow B1 \ldots Bm$.

Other rules in the book.

4^{th} Normal Form (4NF)

R is in 4NF

if for every nontrivial MVD

$A1,\ldots,An \twoheadrightarrow B1,\ldots, Bm,$

{A1,…,An} is a superkey.

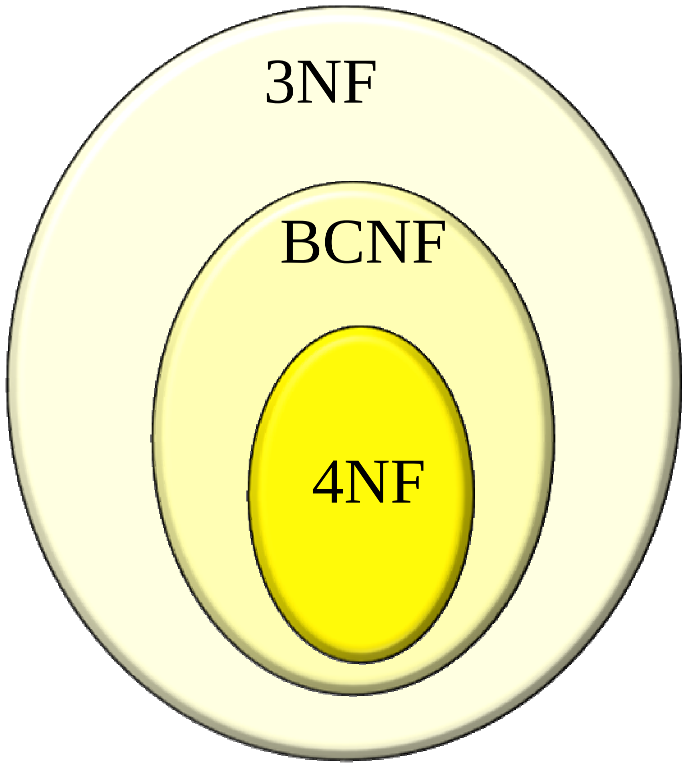
Same as BCNF with FDs replaced by MVDs.

MVD Summary: Parent $\twoheadrightarrow$ Child

- $X \twoheadrightarrow Y$ means that given X, there is a unique *set* of possible Y values (which do not depend on other attributes of the relation)
- MVD problems arise if there are two independent 1:N relationships in a relation.
- An FD is also a MVD.

There's lots more MVD theory, but we won't go there.

Confused by Normal Forms ?



Normal forms tell you when your schema has certain forms of redundancy,
but there is no substitute for commonsense understanding of your application.