Quality test for Relational DBs

Reading: Elmasri & Navathe, Fundamentals of Database Systems, Chapter 14.

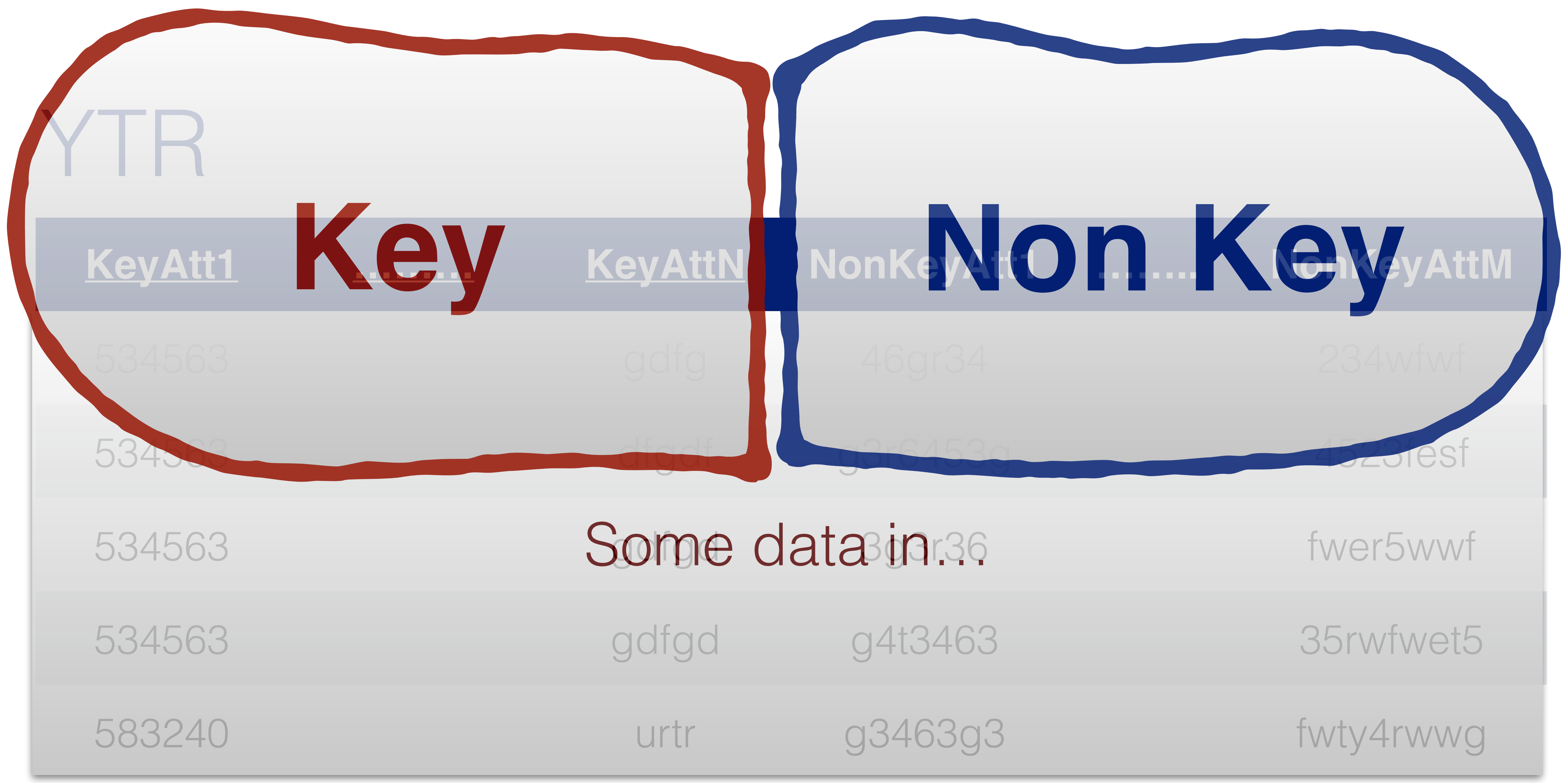
Recap: Normal Forms

- Based on functional dependencies:
    - 1st Normal Form (1NF)
    - 2nd Normal Form (2NF)
    - 3rd Normal Form (3NF)
    - Boyce-Codd Normal Form (BCNF)
- Based on “multivalued” and “join” dependencies:
    - 4th and 5th Normal Forms (4NF, 5NF)
- Each NF extends the previous (so a relation in 3NF is also in 2NF and 1NF)

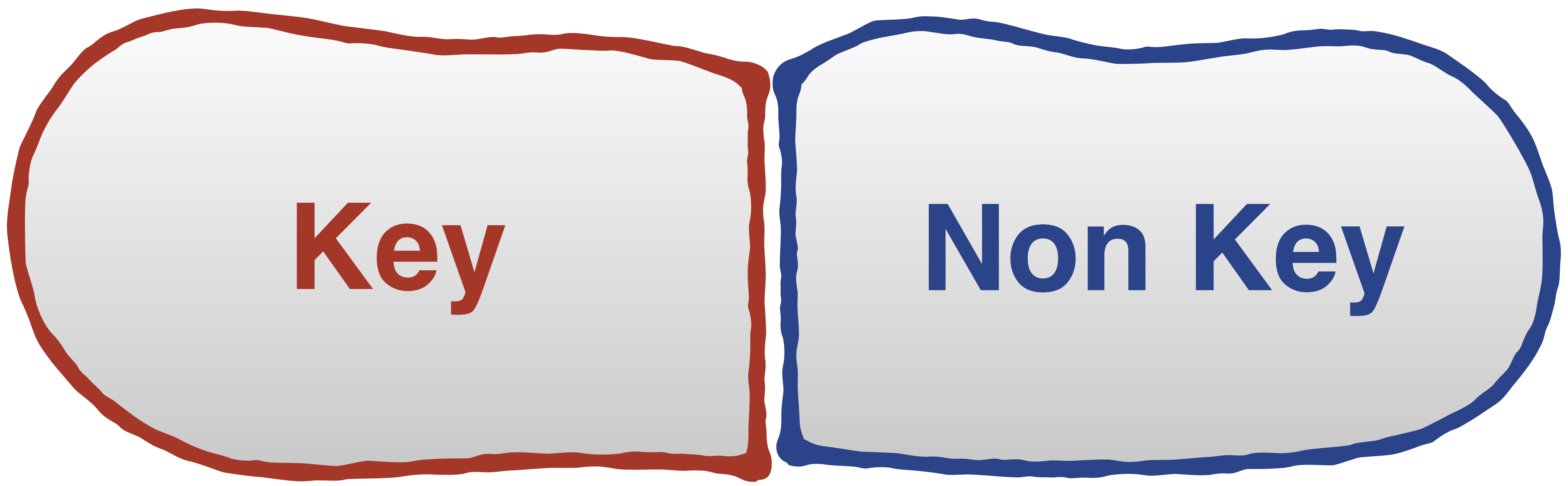
A graphic guide to FD based Normal Forms

consider Your Typical Relation:

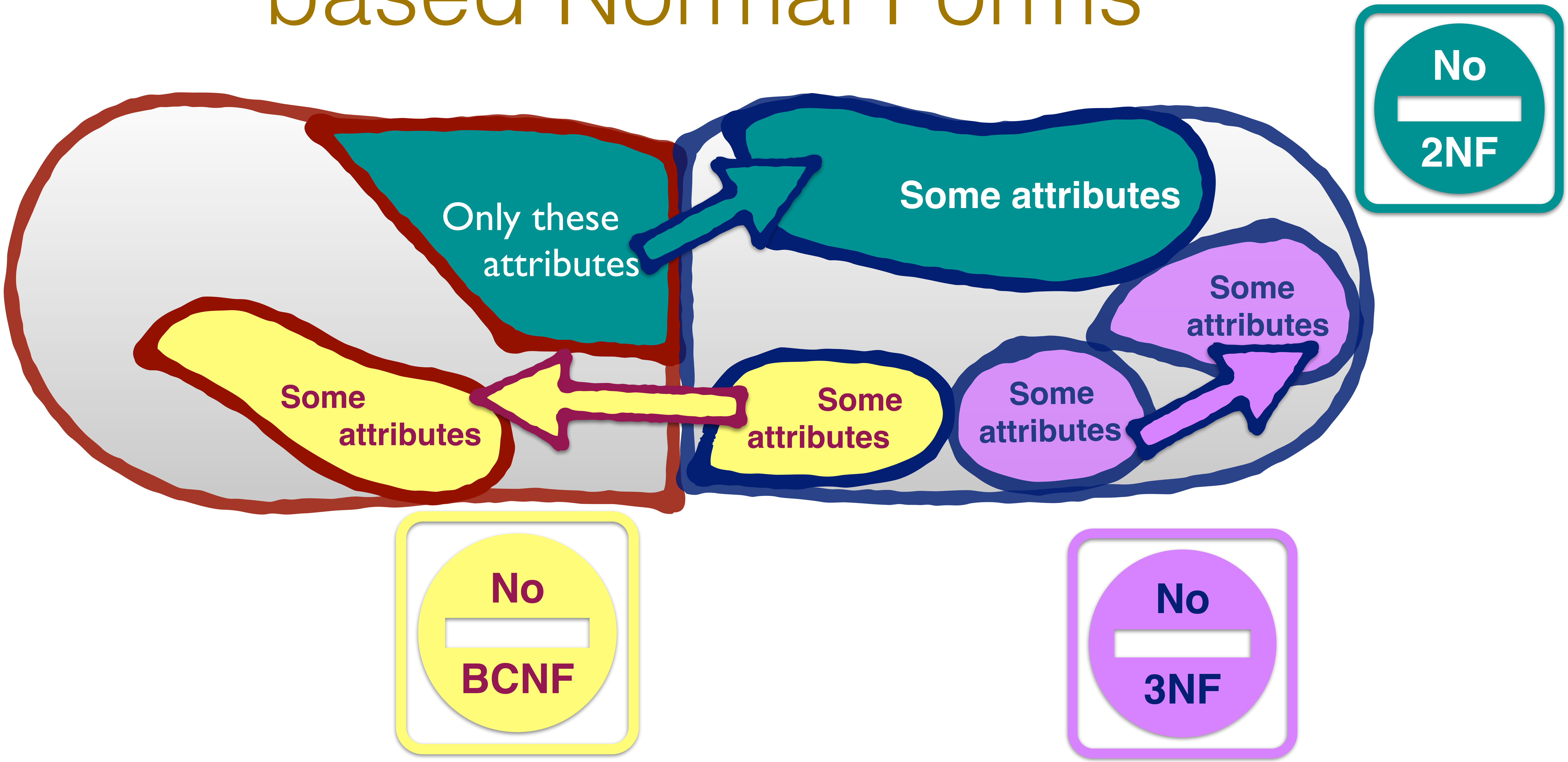
YTR = {***KeyAtt1, KeyAtt2,...KeyAttN***, *NonKeyAtt1,.....NonKeyAttM*}



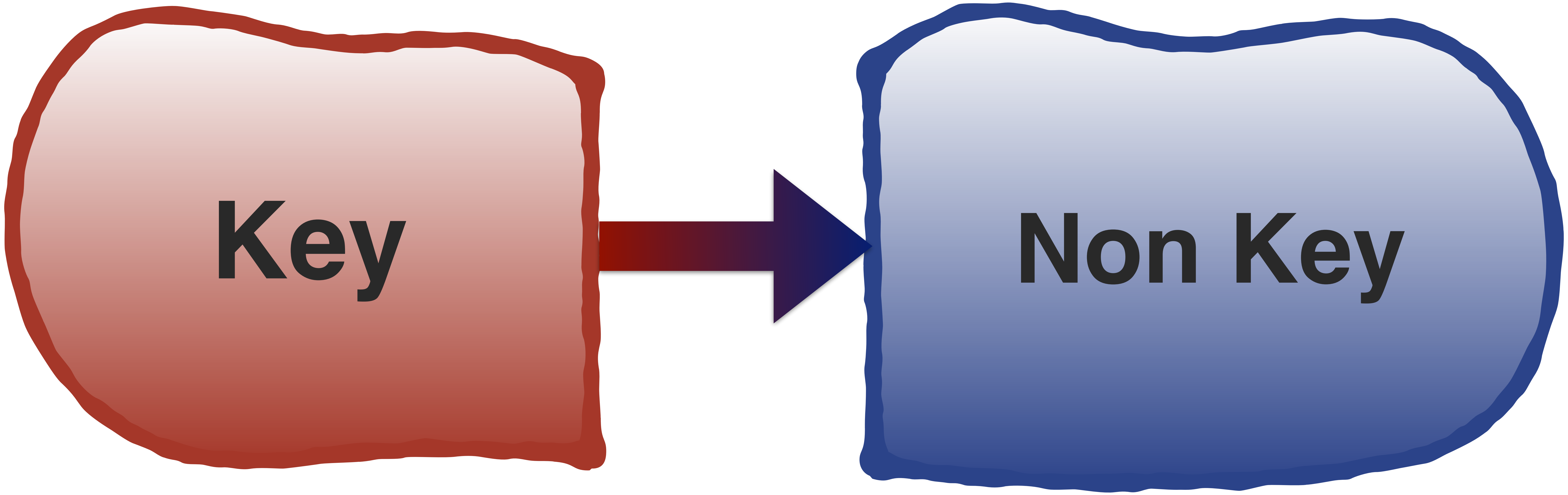
A graphic guide to FD based Normal Forms



A graphic guide to FD based Normal Forms



BCNF (and 3NF, and 2NF)



1. The whole key is needed to determine ALL non key attributes.
2. There are **NO OTHER dependencies.**

Normal Forms: not enough

- When considered in isolation, NFs do not guarantee a good design
- Important additional properties to check:
  - **Dependency preservation property:** Each functional dependency is represented in some relation after normalisation. Desirable but can be sacrificed for other factors.
  - **Non-additive join property:** No spurious tuples are generated after normalisation. Extremely critical.
- Algorithms can be used to guarantee that a set of relations satisfies either or both of the properties above

REMEMBER OUR PROBLEMATIC TABLE?

Students

StudentID	Department	Tutor
123	Physics	Einstein
123	Music	Mozart
456	Biology	Darwin
789	Physics	Bohr
999	Physics	Einstein

- this was non BCNF
- 3 alternative solutions
- let's test them all

1.

StudentID	Tutor
123	Einstein
123	Mozart
456	Darwin
789	Bohr
999	Einstein

StudentID	Department
123	Physics
123	Music
456	Biology
789	Physics
999	Physics

2.

Tutor	Department
Einstein	Physics
Mozart	Music
Darwin	Biology
Bohr	Physics

StudentID	Department
123	Physics
123	Music
456	Biology
789	Physics
999	Physics

3.

Tutor	Department
Einstein	Physics
Mozart	Music
Darwin	Biology
Bohr	Physics

StudentID	Tutor
123	Einstein
123	Mozart
456	Darwin
789	Bohr
999	Einstein

Dependency preservation

Students

StudentID	Department	Tutor
123	Physics	Einstein
123	Music	Mozart
456	Biology	Darwin
789	Physics	Bohr
999	Physics	Einstein

- Functional Dependencies:
  1. *{StudentID, Department} → Tutor*
  2. *Tutor → Department*

3.

Tutor	Department
Einstein	Physics
Mozart	Music
Darwin	Biology
Bohr	Physics

StudentID	Tutor
123	Einstein
123	Mozart
456	Darwin
789	Bohr
999	Einstein

- This set of tables is BCNF
- However: Functional Dependency 1. is **not** preserved
- There is no way we can tell it holds

Non-additive join

1.

StudentID	Tutor
123	Einstein
123	Mozart
456	Darwin
789	Bohr
999	Einstein

StudentID	Department
123	Physics
123	Music
456	Biology
789	Physics
999	Physics

This BCNF decomposition does not have the **non-additive join property** as spurious tuples can be produced

R1 join R2 =

StudentID	Department	Tutor
123	Physics	Einstein
123	Music	Mozart
123	Physics	Mozart
123	Music	Einstein
456	Biology	Darwin
789	Physics	Bohr
999	Physics	Einstein

≠

Students

StudentID	Department	Tutor
123	Physics	Einstein
123	Music	Mozart
456	Biology	Darwin
789	Physics	Bohr
999	Physics	Einstein

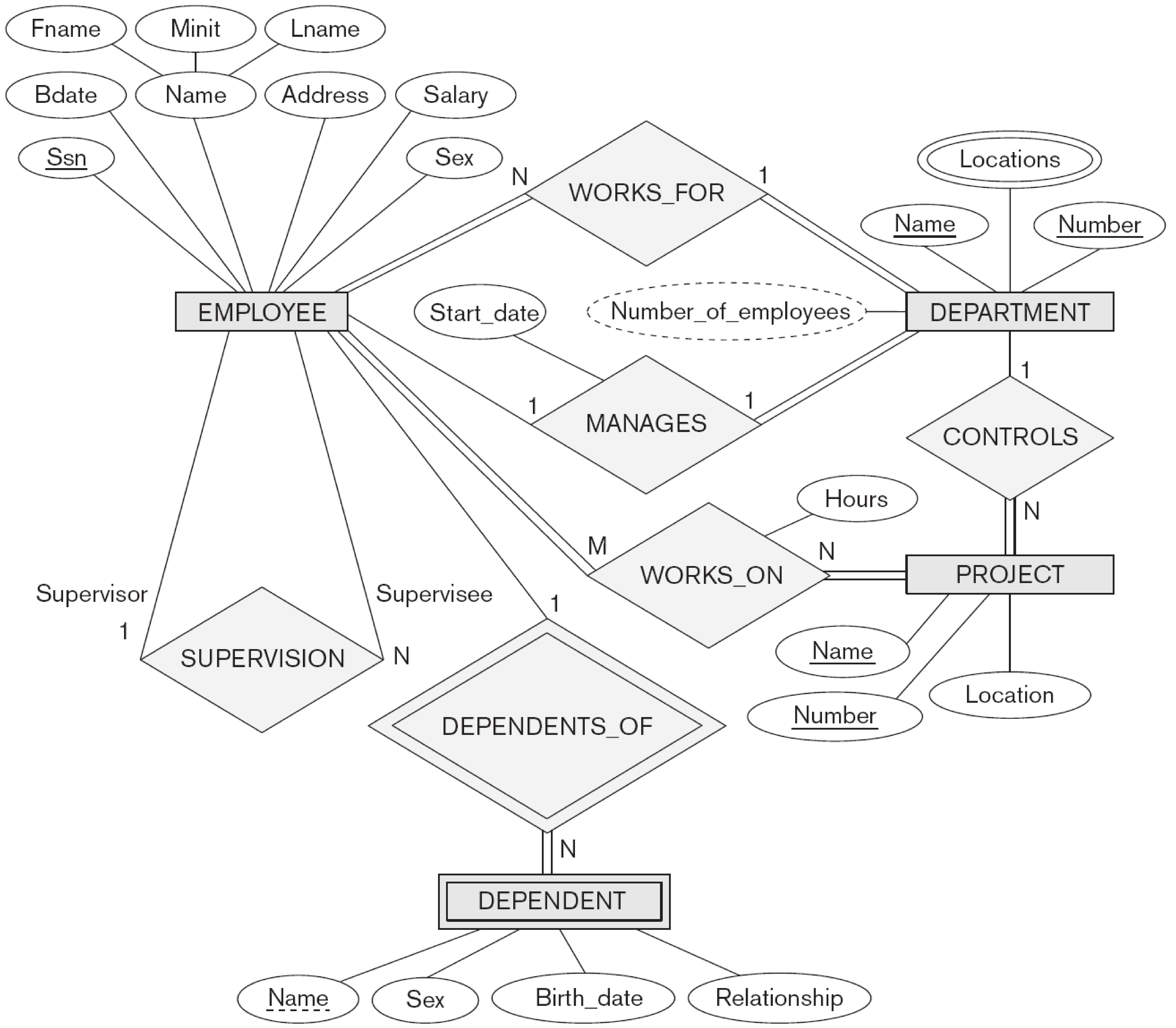
Testing for non-additive join

- Problem: relying on finding examples is not the best strategy
- We need a more systematic procedure
- **“Tableaux test for non-additive join”**
    - starting from the "universal relation” and the set of FDs
    - gives an algorithm for easy checking
    - if the test succeeds, we can be sure ALL instances of the database will never generate spurious tuples
    - if the test fails, it also gives the counterexample we need

Tableaux test

- Principle: simulate what would happen if a report was created that joined *all tables* in the database
- One would assume that only very few individuals are such that their relationships can be traced through *all* tables
- If a join over all tables produced a potentially infinite amount of such individuals, we would be in the presence of spurious records
- We would like to test this, without having to populate the database with 1000s of records in the hope of finding a good example by chance

In the company example, this would mean to search for all employees **who** work for a department, **and** manage a department, **and** work on a project, **which is** controlled by a department, **and** have a supervisor, **and** have a dependent.



We can simulate this by building a table containing **ALL** attributes of the database (the "Universal Relation") and **one row for each** table (relation) of the DB, and filling it in with generic values, arbitrary, **but constrained by the functional dependencies**

	Attribute 1	Attribute 2	…	…	…	Attribute N
Relation 1						
Relation 2						
…						
…						
Relation M						

Let’s test our example

3.

Tutor	Department
Einstein	Physics
Mozart	Music
Darwin	Biology
Bohr	Physics

StudentID	Tutor
123	Einstein
123	Mozart
456	Darwin
789	Bohr
999	Einstein

- The Universal relation is {StudentID, Tutor, Department}
- The Relations are R1={Tutor, Department} and R2={StudentID,Tutor}
- The only functional dependency that we can represent here is Tutor → Department

- Universal relation {StudentID, Tutor, Department}
- R1={Tutor, Department} and R2={StudentID,Tutor}
- FD: Tutor → Department

1. Prepare the table:

	StudentID	Tutor	Department
R_1			
R_2			

- Universal relation {StudentID, Tutor, Department}
- R1={Tutor, Department} and R2={StudentID,Tutor}
- FD: Tutor → Department

2. Fill each row with generic values, ensuring consistency:

We use each relation in turn and add values to "activated" attributes in each row

	StudentID	Tutor	Department
R_1		Value 1	Value 2
R_2			

- Universal relation {StudentID, Tutor, Department}
- R1={Tutor, Department} and
  R2={StudentID,Tutor}
- FD: Tutor → Department

2. Fill each row with generic values, ensuring consistency:

We use each relation in turn and add values to "activated" attributes in each row

	StudentID	Tutor	Department
R_1		Value 1	Value 2
R_2	Value 3	Value 1	

- Universal relation {StudentID, Tutor, Department}
- R1={Tutor, Department} and
  R2={StudentID,Tutor}
- FD: Tutor → Department

2. Fill each row with generic values, ensuring consistency:

We use each relation in turn and add values to "activated" attributes in each row

	StudentID	Tutor	Department
R_1		Value 1	Value 2
R_2	Value 3	Value 1	

- Universal relation {StudentID, Tutor, Department}
- R1={Tutor, Department} and R2={StudentID,Tutor}
- FD: Tutor → Department

"All tuples with same Tutor must have the same Department"

3. Apply the Functional Dependencies in turn

	StudentID	Tutor	Department
R_1		Value 1	Value 2
R_2	Value 3	Value 1	Value 2

- Universal relation {StudentID, Tutor, Department}
- R1={Tutor, Department} and
  R2={StudentID,Tutor}
- FD: Tutor → Department

If we manage to complete a row of values, we pass the test!

	StudentID	Tutor	Department
R_1		Value 1	Value 2
R_2	Value 3	Value 1	Value 2

- Universal relation {StudentID, Tutor, Department}
- R1={Tutor, Department} and
  R2={StudentID,Tutor}
- FD: Tutor → Department

If we manage to complete a row of values, we pass the test!

This DB is safe from producing spurious records ✔ 👍

	StudentID	Tutor	Department
R_1		Value 1	Value 2
R_2	Value 3	Value 1	Value 2

Let's test another example

2.

Tutor	Department
Einstein	Physics
Mozart	Music
Darwin	Biology
Bohr	Physics

StudentID	Department
123	Physics
123	Music
456	Biology
789	Physics
999	Physics

- The Universal relation is {StudentID, Tutor, Department}
- The Relations are R1={Tutor, Department} and R2={StudentID,Department}
- The functional dependency that we can represent here is only: Tutor → Department

- Universal relation {StudentID, Tutor, Department}
- R1={Tutor, Department} and R2={StudentID,Department}
- FD: Tutor → Department

1. Prepare the table:

	StudentID	Tutor	Department
R_1			
R_2			

- Universal relation {StudentID, Tutor, Department}
- $R1$={Tutor, Department} and
  $R2$={StudentID,Department}
- FD: Tutor → Department

2. Fill each row with generic values, ensuring consistency:

	StudentID	Tutor	Department
R_1		Value 1	Value 2
R_2			

- Universal relation {StudentID, Tutor, Department}
- R1={Tutor, Department} and R2={StudentID,Department}
- FD: Tutor → Department

2. Fill each row with generic values, ensuring consistency:

	StudentID	Tutor	Department
R_1		Value 1	Value 2
R_2	Value 3		Value 2

- Universal relation {StudentID, Tutor, Department}
- R1={Tutor, Department} and R2={StudentID,Department}
- FD: Tutor → Department

2. Fill each row with generic values, ensuring consistency:

	StudentID	Tutor	Department
R_1		Value 1	Value 2
R_2	Value 3		Value 2

- Universal relation {StudentID, Tutor, Department}
- R1={Tutor, Department} and R2={StudentID,Department}
- FD: Tutor → Department

"All tuples with same Tutor must have the same Department"

3. Apply the Functional Dependencies in turn

	StudentID	Tutor	Department
R_1		Value 1	Value 2
R_2	Value 3	✗	Value 2

- Universal relation {StudentID, Tutor, Department}
- R1={Tutor, Department} and R2={StudentID,Department}
- FD: Tutor ~~→~~ Department

There are no 2 tuples where we can apply this FD

3. Apply the Functional Dependencies in turn

	StudentID	Tutor	Department
R_1		Value 1	Value 2
R_2	Value 3		Value 2

- Universal relation {StudentID, Tutor, Department}
- R1={Tutor, Department} and R2={StudentID,Department}
- FD: Tutor ~~→~~ Department

There are no 2 tuples where we can apply this FD

If we manage to complete a row of values, we pass the test!

This DB could produce spurious records ✗

	StudentID	Tutor	Department
R_1		Value 1	Value 2
R_2	Value 3		Value 2

The final table can be used as counterexample: let's fill it in with more meaningful values

	StudentID	Tutor	Department
R_1		Value 1	Value 2
R_2	Value 3		Value 2

The final table can be used as counterexample: let's fill it in with more meaningful values

	StudentID	Tutor	Department
R_1		Value 1	CompSci
R_2	Value 3		CompSci

The final table can be used as counterexample: let's fill it in with more meaningful values

	StudentID	Tutor	Department
R_1		Turing	CompSci
R_2	Value 3	Hopper	CompSci

The final table can be used as counterexample: let's fill it in with more meaningful values

StudentID	Tutor	Department
St1	Turing	CompSci
St2	Hopper	CompSci

R1={Tutor, Department}

Tutor	Department
Turing	CompSci
Hopper	CompSci

R2={StudentID,Department}

StudentID	Department
St1	CompSci
St2	CompSci

StudentID	Department
St1	CompSci
St2	CompSci

$\bowtie$

Tutor	Department
Turing	CompSci
Hopper	CompSci

=

StudentID	Department	Tutor
St1	CompSci	Turing
St2	CompSci ✗	Turing
St1	CompSci	Hopper
St2	CompSci	Hopper

$\neq$

StudentID	Tutor	Department
St1	Turing	CompSci
St2	Hopper	CompSci

What about solution 1?

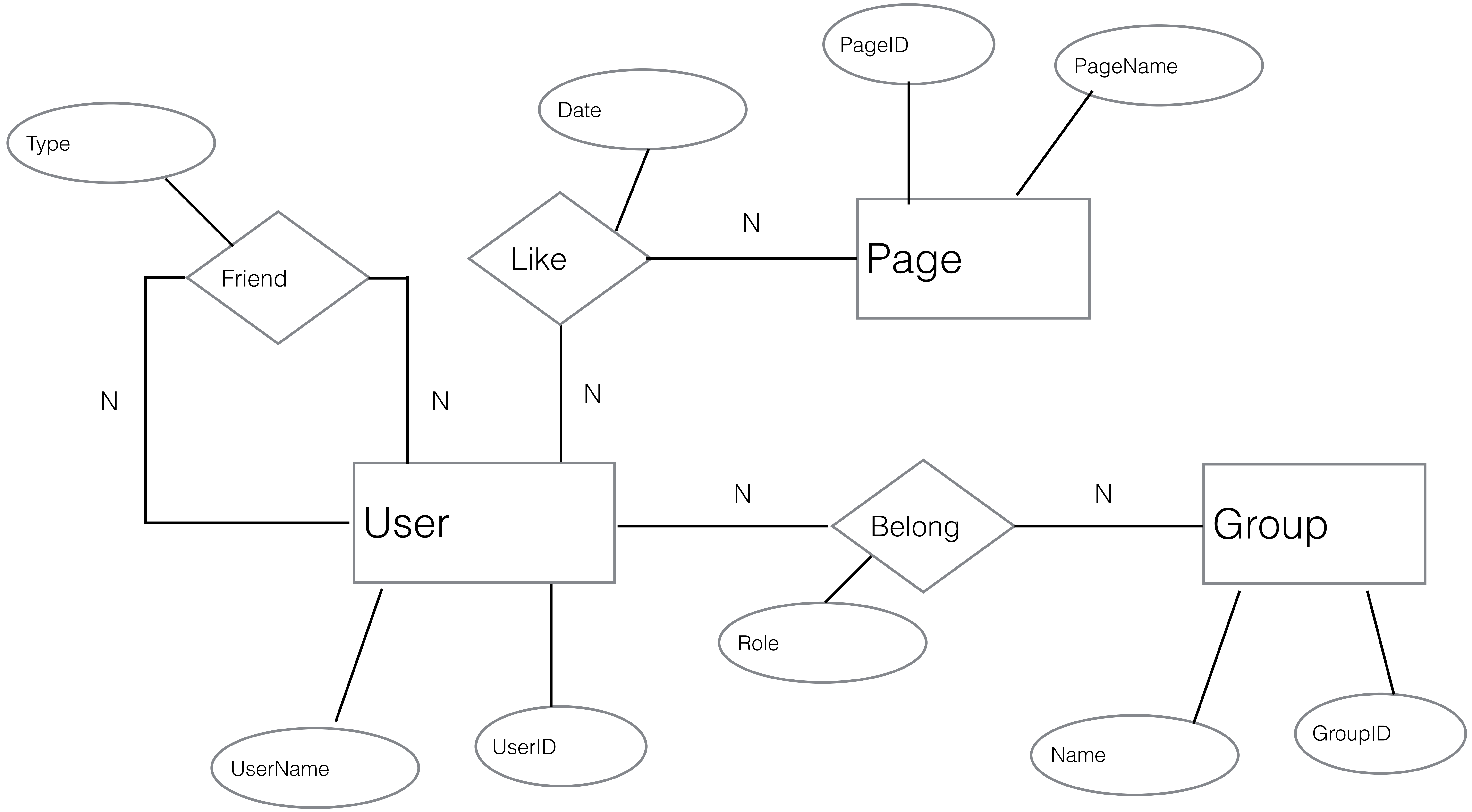
1.

StudentID	Tutor
123	Einstein
123	Mozart
456	Darwin
789	Bohr
999	Einstein

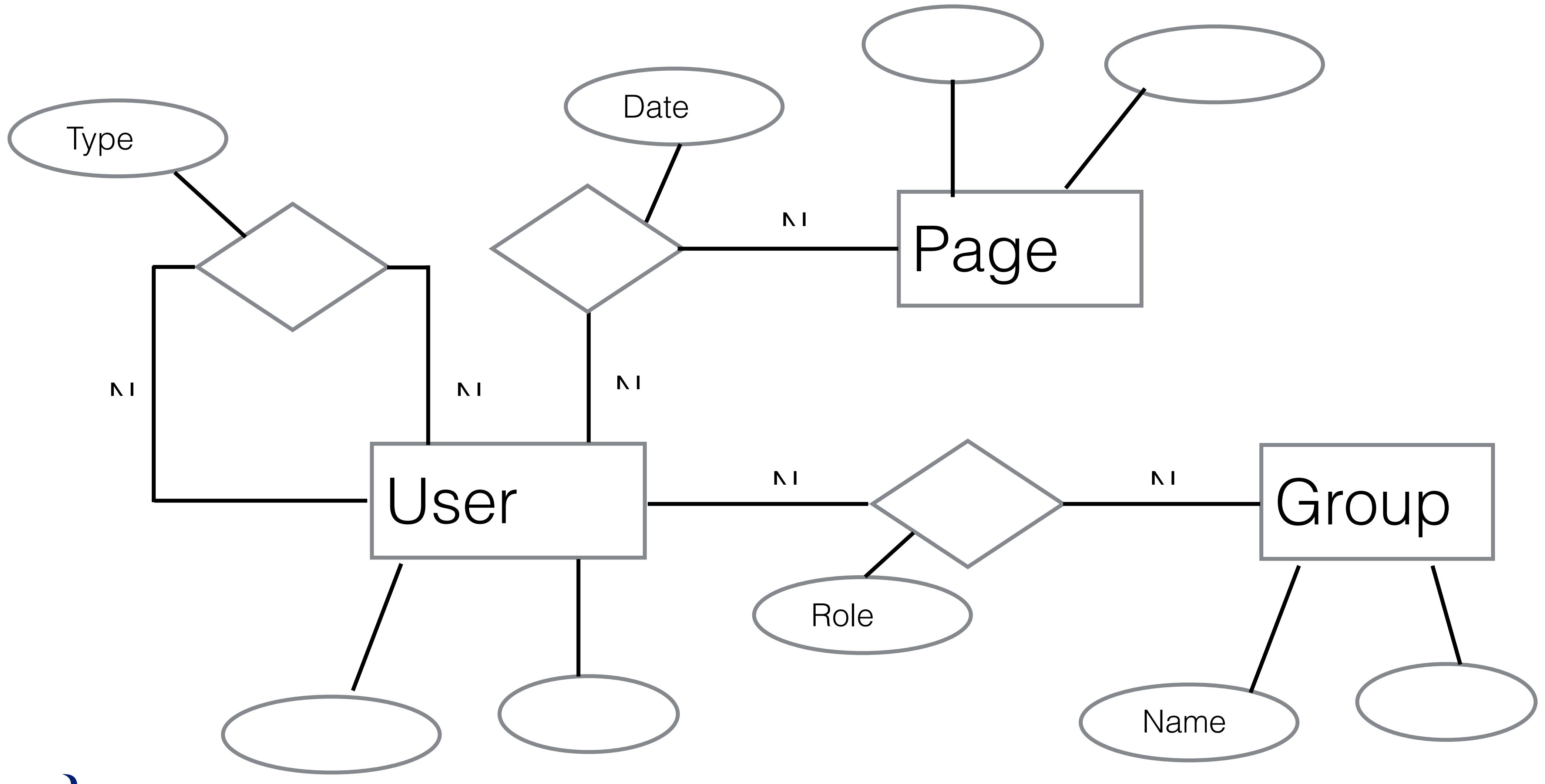
StudentID	Department
123	Physics
123	Music
456	Biology
789	Physics
999	Physics

- The Universal relation is {StudentID, Tutor, Department}
- The Relations are R1={StudentID, Tutor} and R2={StudentID,Department}
- The functional dependencies that we can represent here are…
  - NONE!
- The decomposition fails the test immediately

No design is waterproof!

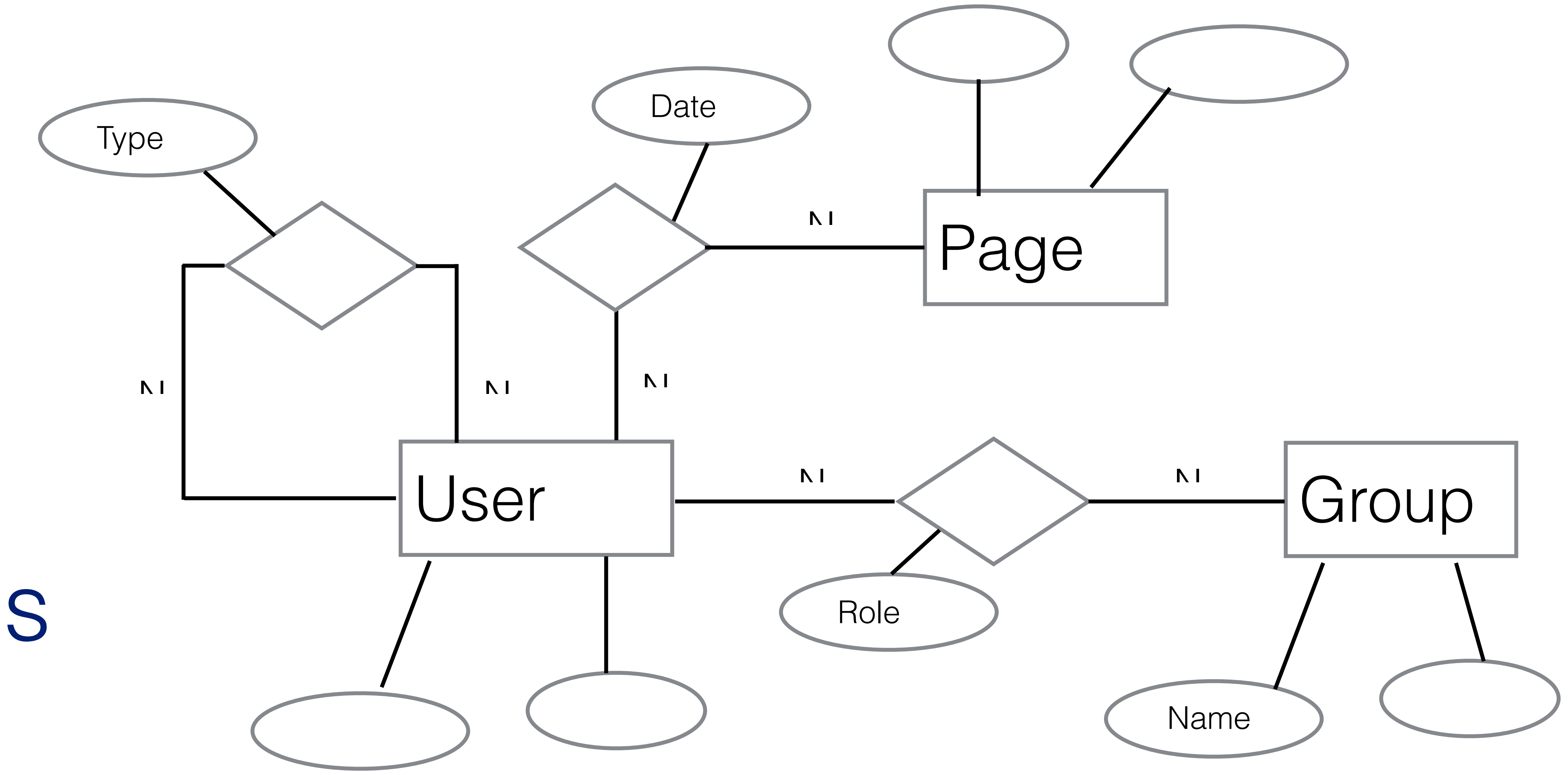


A simplified social network!



Relations:

- R1={UserID, Username}
- R2={GroupID, GroupName}
- R3={PageID, PageName}
- R4={UserID,PageID,Date} (*user likes page*)
- R5={UserID,GroupID,Role} (*user belongs to group*)
- R6={UserID,FriendID,Type} (*user friends user*)



Functional dependencies

- *UserID → Username*
- *GroupID → GroupName*
- *PageID → PageName*
- *{UserID,PageID} → Date*
- *{UserID,GroupID} → Role*
- *{UserID,FriendID} → Type*

- BCNF set of tables
- and we also can preserve all dependencies

The Tableaux test

- Construct the rows, one for each relation:

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	Value1	Value2								
R2			Value3	Value4						
R3					Value5	Value6				
R4	Value1				Value5		Value7			
R5	Value1		Value3					Value8		
R6	Value1								Value9	Value10

The Tableaux test

- *UserID → Username*

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	Value1	Value2								
R2			Value3	Value4						
R3					Value5	Value6				
R4	Value1				Value5		Value7			
R5	Value1		Value3					Value8		
R6	Value1								Value9	Value10

The Tableaux test

- *UserID → Username*

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	Value1	Value2								
R2			Value3	Value4						
R3					Value5	Value6				
R4	Value1	Value2			Value5		Value7			
R5	Value1	Value2	Value3					Value8		
R6	Value1	Value2							Value9	Value10

The Tableaux test

- *GroupID → GroupName*

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	Value1	Value2								
R2			Value3	Value4						
R3					Value5	Value6				
R4	Value1	Value2			Value5		Value7			
R5	Value1	Value2	Value3					Value8		
R6	Value1	Value2							Value9	Value10

The Tableaux test

- *GroupID → GroupName*

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	Value1	Value2								
R2			Value3	Value4						
R3					Value5	Value6				
R4	Value1	Value2			Value5		Value7			
R5	Value1	Value2	Value3	Value4				Value8		
R6	Value1	Value2							Value9	Value10

The Tableaux test

- *PageID → PageName*

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	Value1	Value2								
R2			Value3	Value4						
R3					Value5	Value6				
R4	Value1	Value2			Value5		Value7			
R5	Value1	Value2	Value3	Value4				Value8		
R6	Value1	Value2							Value9	Value10

The Tableaux test

- *PageID → PageName*

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	Value1	Value2								
R2			Value3	Value4						
R3					Value5	Value6				
R4	Value1	Value2			Value5	Value6	Value7			
R5	Value1	Value2	Value3	Value4				Value8		
R6	Value1	Value2							Value9	Value10

The Tableaux test

- *{UserID,PageID} → Date*

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	Value1	Value2			X					
R2			Value3	Value4						
R3	X				Value5	Value6				
R4	Value1	Value2			Value5	Value6	Value7			
R5	Value1	Value2	Value3	Value4	X			Value8		
R6	Value1	Value2			X				Value9	Value10

The Tableaux test

- *{UserID, GroupID} → Role*

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	Value1	Value2	X							
R2	X		Value3	Value4						
R3					Value5	Value6				
R4	Value1	Value2	X		Value5	Value6	Value7			
R5	Value1	Value2	Value3	Value4				Value8		
R6	Value1	Value2	X						Value9	Value10

The Tableaux test

- *{UserID,FriendID} → Type*

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	Value1	Value2							✗	
R2			Value3	Value4						
R3					Value5	Value6				
R4	Value1	Value2			Value5	Value6	Value7		✗	
R5	Value1	Value2	Value3	Value4				Value8	✗	
R6	Value1	Value2							Value9	Value10

The Tableaux test

This DB could produce spurious records!!!

✗

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	Value1	Value2								
R2			Value3	Value4						
R3					Value5	Value6				
R4	Value1	Value2			Value5	Value6	Value7			
R5	Value1	Value2	Value3	Value4				Value8		
R6	Value1	Value2							Value9	Value10

The counterexample

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	Value1	Value2								
R2			Value3	Value4						
R3					Value5	Value6				
R4	Value1	Value2			Value5	Value6	Value7			
R5	Value1	Value2	Value3	Value4				Value8		
R6	Value1	Value2							Value9	Value10

The counterexample

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	U1	John							..	..
R2			G1	Bikers					..	..
R3					P1	StarWars			..	..
R4	U1	John			P1	StarWars	1.4.19		..	..
R5	U1	John	G1	Bikers				Treasurer	..	..
R6	U1	John							..	...

Ignore this for brevity

The counterexample

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	U1	John	G2	Geeks	P2	Pizza	1.1.19	Admin	..	..
R2	U2	Mary	G1	Bikers	P3	Liverpool	1.2.19	Owner	..	..
R3	U3	Sarah	G3	Chefs	P1	StarWars	1.3.19	Fan	..	..
R4	U1	John	G4	Travel	P1	StarWars	1.4.19	Supporter	..	..
R5	U1	John	G1	Bikers	P4	Van Gogh	1.5.19	Treasurer	..	..
R6	U1	John	G5	MovieClub	P5	Java!	1.6.19	Chief	...	...

Ignore this for brevity

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	Frie	Type
R1	U1	John	G2	Geeks	P2	Pizza	1.1.19	Admin	..	..
R2	U2	Mary	G1	Bikers	P3	Liverpool	1.2.19	Owner	..	..
R3	U3	Sarah	G3	Chefs	P1	StarWars	1.3.19	Fan	..	..
R4	U1	John	G4	Travel	P1	StarWars	1.4.19	Supporter	..	..
R5	U1	John	G1	Bikers	P4	Van Gogh	1.5.19	Treasurer	..	..
R6	U1	John	G5	MovieClub	P5	Java!	1.6.19	Chief	…	…

UserID	Username
U1	John
U2	Mary
U3	Sarah

GroupID	GroupName
G1	Bikers
G2	Geeks
G3	Chefs
G4	Travel
G5	MovieClub

PageID	PageName
P1	StarWars
P2	Pizza
P3	Liverpool
P4	Van Gogh
P5	Java!

UserID	GroupID	Role
U1	G2	Admin
U2	G1	Owner
U3	G3	Fan
U1	G4	Supporter
U1	G1	Treasurer
U1	G5	Chief

UserID	PageID	Date
U1	P2	1.1.19
U2	P3	1.2.19
U3	P1	1.3.19
U1	P1	1.4.19
U1	P4	1.5.19
U1	P5	1.6.19

- looks reasonable!
  - what you would expect
- BUT!

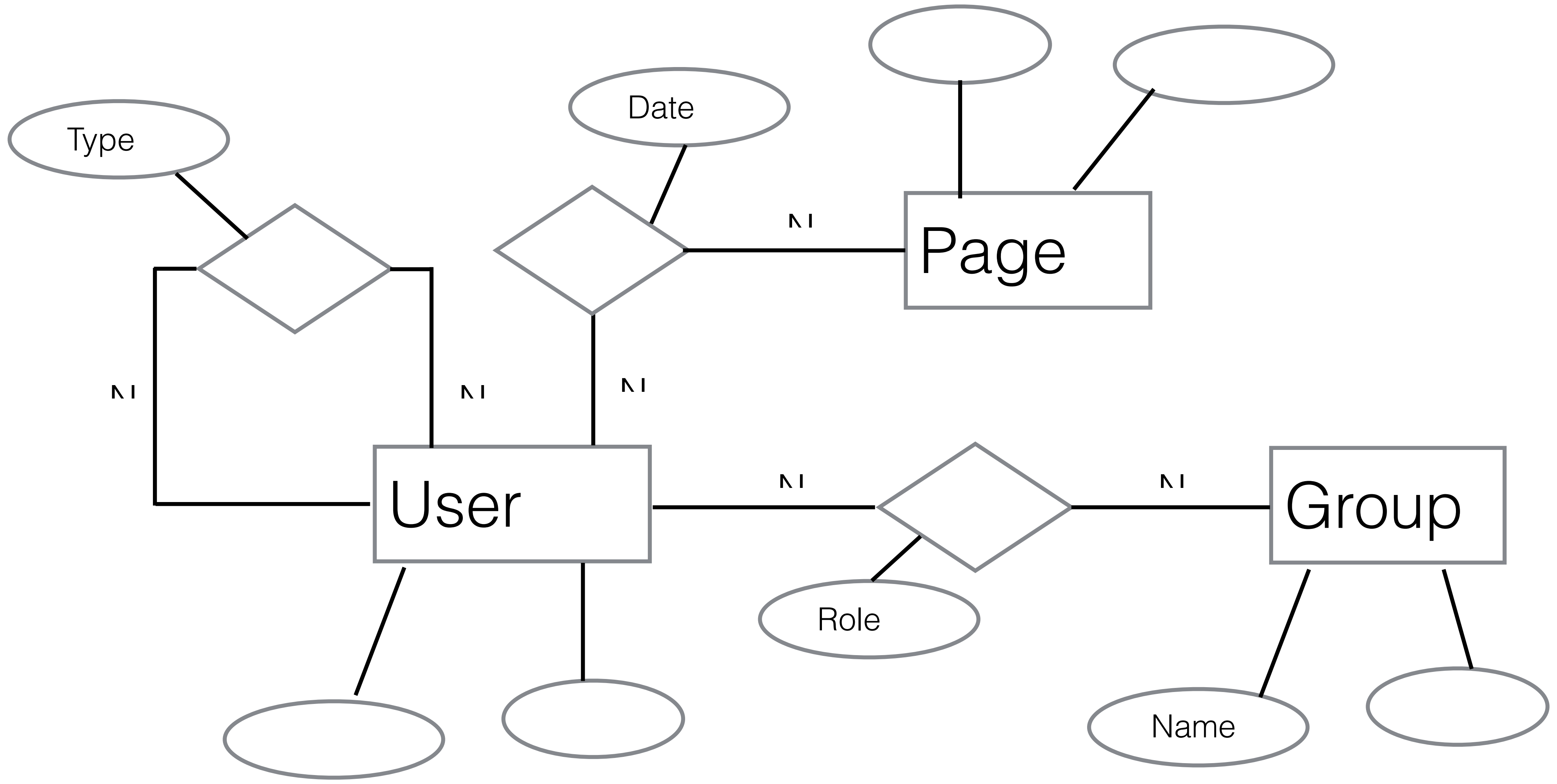
UserID
U1
U2
U3

UserID	Username	PageID	Date	PageName	GroupID	Role	GroupName
U1	John	P2	1.1.19	Pizza	G2	ADMIN	Geeks
U1	John	P2	1.1.19	Pizza	G4	SUPPORTER	Travel
U1	John	P2	1.1.19	Pizza	G1	TREASURER	Bikers
U1	John	P2	1.1.19	Pizza	G5	CHIEF	MovieClub
U1	John	P1	1.4.19	StarWars	G2	ADMIN	Geeks
U1	John	P1	1.4.19	StarWars	G4	SUPPORTER	Travel
U1	John	P1	1.4.19	StarWars	G1	TREASURER	Bikers
U1	John	P1	1.4.19	StarWars	G5	CHIEF	MovieClub
U1	John	P4	1.5.19	VanGogh	G2	ADMIN	Geeks
U1	John	P4	1.5.19	VanGogh	G4	SUPPORTER	Travel
U1	John	P4	1.5.19	VanGogh	G1	TREASURER	Bikers
U1	John	P4	1.5.19	VanGogh	G5	CHIEF	MovieClub
U1	John	P5	1.6.19	Java	G2	ADMIN	Geeks
U1	John	P5	1.6.19	Java	G4	SUPPORTER	Travel
U1	John	P5	1.6.19	Java	G1	TREASURER	Bikers
U1	John	P5	1.6.19	Java	G5	CHIEF	MovieClub
U2	Mary	P3	1.2.19	Liverpool	G1	OWNER	Bikers
U3	Sarah	P1	1.3.19	StarWars	G3	FAN	Chefs

!

We are stuck then?

- NO, there is a simple trick that can *fix* any database
- Always add a “bridge relation” to your set of tables
- This is a relation that contains the “superkey” of the whole database, i.e. the key of the Universal Relation
- This will capture the individuals that “tick all boxes”



Relations:

- R1={UserID, Username}
- R2={GroupID, GroupName}
- R3={PageID, PageName}
- R4={UserID,PageID,Date} (*user likes page*)
- R5={UserID,GroupID,Role} (*user belongs to group*)
- R6={UserID,FriendID,Type} (*user friends user*)
- **R7={UserID,GroupID,PageID,FriendID}   superkey**

Updated Tableaux test

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	Value1	Value2								
R2			Value3	Value4						
R3					Value5	Value6				
R4	Value1				Value3		Value7			
R5	Value1		Value3					Value8		
R6	Value1								Value9	Value10
R7	Value1		Value3		Value3				Value9	

Updated Tableaux test

✔ 👍

The bridge R7 will always guarantee the test is passed

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	Value1	Value2								
R2			Value3	Value4						
R3					Value5	Value6				
R4	Value1	Value2			Value5	Value6	Value7			
R5	Value1	Value2	Value3	Value4				Value8		
R6	Value1	Value2							Value9	Value10
R7	Value1	Value2	Value3	Value4	Value5	Value6	Value7	Value8	Value9	Value10

Updated counterexample

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	FriendID	Type
R1	U1	John	G2	Geeks	P2	Pizza	1.1.19	Admin	..	..
R2	U2	Mary	G1	Bikers	P3	Liverpool	1.2.19	Owner	..	..
R3	U3	Sarah	G3	Chefs	P1	StarWars	1.3.19	Fan	..	..
R4	U1	John	G4	Travel	P1	StarWars	1.4.19	Supporter	..	..
R5	U1	John	G1	Bikers	P4	Van Gogh	1.5.19	Treasurer	..	..
R6	U1	John	G5	MovieClub	P5	Java!	1.6.19	Chief	…	…
R7	U1	John	G1	Bikers	P1	StarWars	1.4.19	Treasurer		

Ignore this for brevity

	UserID	Username	GroupID	GroupName	PageID	PageName	Date	Role	Frie	Type
R1	U1	John	G2	Geeks	P2	Pizza	1.1.19	Admin	..	..
R2	U2	Mary	G1	Bikers	P3	Liverpool	1.2.19	Owner	..	..
R3	U3	Sarah	G3	Chefs	P1	StarWars	1.3.19	Fan	..	..
R4	U1	John	G4	Travel	P1	StarWars	1.4.19	Supporter	..	..
R5	U1	John	G1	Bikers	P4	Van Gogh	1.5.19	Treasurer	..	..
R6	U1	John	G5	MovieClub	P5	Java!	1.6.19	Chief	…	…
R7	U1	John	G1	Bikers	P1	StarWars	1.4.19	Treasurer		

UserID	Username
U1	John
U2	Mary
U3	Sarah

GroupID	GroupName
G1	Bikers
G2	Geeks
G3	Chefs
G4	Travel
G5	MovieClub

PageID	PageName
P1	StarWars
P2	Pizza
P3	Liverpool
P4	Van Gogh
P5	Java!

These are the users who both have a page and belong to a group, in all legal combinations

UserID	GroupID	Role
U1	G2	Admin
U2	G1	Owner
U3	G3	Fan
U1	G4	Supporter
U1	G1	Treasurer
U1	G5	Chief

UserID	PageID	Date
U1	P2	1.1.19
U2	P3	1.2.19
U3	P1	1.3.19
U1	P1	1.4.19
U1	P4	1.5.19
U1	P5	1.6.19

UserID	GroupID	PageID
U1	G2	P2
U2	G1	P3
U3	G3	P1
U1	G4	P1
U1	G1	P4
U1	G5	P5
U1	G1	P1

UserID
U1
U2
U3

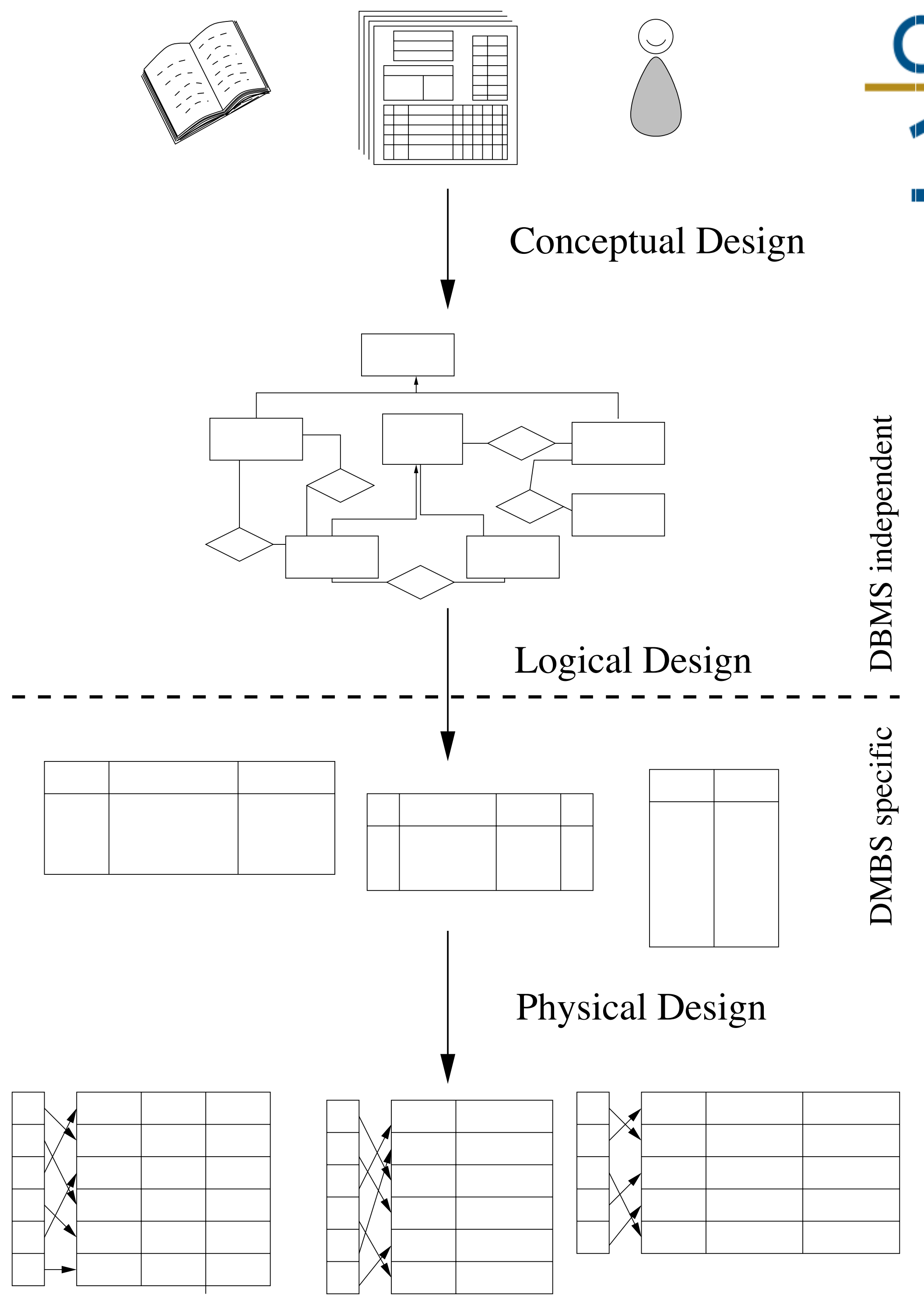
$\bowtie$

UserID	Username	PageID	Date	PageName	GroupID	Role	GroupName
U1	John	P2	1.1.19	Pizza	G2	ADMIN	Geeks
U1	John	P1	1.4.19	StarWars	G4	SUPPORTER	Travel
U1	John	P1	1.4.19	StarWars	G1	TREASURER	Bikers
U1	John	P4	1.5.19	VanGogh	G1	TREASURER	Bikers
U1	John	P5	1.6.19	Java	G5	CHIEF	MovieClub
U2	Mary	P3	1.2.19	Liverpool	G1	OWNER	Bikers
U3	Sarah	P1	1.3.19	StarWars	G3	FAN	Chefs

=

U1	G1	P1

Conclusions



- We have gone full cycle (design wise at least)
    - from requirements to user stories
    - from user stories to conceptual design, in (E)ER
    - from conceptual design to logical design (in the relational model)
- we can identify issues and critical points in each of the phases, and we can express all of the above in a formal report
- Ready to go back to a more general notion of *quality* in CS/IT