Behind the Scene: Know whom we should blame?



Normal form	Defined by	Brief definition
First normal form (1NF)	Two versions: E.F. Codd (1970), C.J. Date (2003)[12]	Table faithfully represents a relation and has no "repeating groups"
Second normal form (2NF)	E.F. Codd (1971)[13]	No non-prime attribute in the table is functionally dependent on a part (proper subset) of a candidate key
Third normal form (3NF)	E.F. Codd (1971)[14]; see also Carlo Zaniolo's equivalent but differently-expressed definition (1982)[15]	Every non-prime attribute is non-transitively dependent on every key of the table
Boyce-Codd normal form (BCNF)	Raymond F. Boyce and E.F. Codd (1974)[16]	Every non-trivial functional dependency in the table is a dependency on a superkey
Fourth normal form (4NF)	Ronald Fagin (1977)[17]	Every non-trivial multivalued dependency in the table is a dependency on a superkey
Fifth normal form (5NF)	Ronald Fagin (1979)[18]	Every non-trivial join dependency in the table is implied by the superkeys of the table
Domain/key normal form (DKNF)	Ronald Fagin (1981)[19]	Every constraint on the table is a logical consequence of the table's domain constraints and key constraints
Sixth normal form (6NF)	Chris Date, Hugh Darwen, and Nikos Lorentzos (2002)[20]	Table features no non-trivial join dependencies at all (with reference to generalized join operator)

Our Attack Plan

- Motivation
- Functional dependencies & keys
- Reasoning with FDs and keys
- Desirable properties of schema refinement
- Various normal forms and the trade-offs
  - BCNF, 3rd normal form, 4th normal form, etc.
- Putting all together: how to design DB schema

Functional Dependencies and Keys

Better Designs Exist

Break the relation into two:

	SSN	Address
	123-321-99	10 Addr
Alex	123-321-99	10 Green
Sally	909-438-44	431 Purple

X

Identify

?

Address X

SSN	Phone Number	
123-321-99	(201) 555-1234	A1
123-321-99	(206) 572-4312	A1
909-438-44	(908) 464-0028	A2
909-438-44	(212) 555-4000	A2

Reminder

Tutorial #2.

→ Today! 4:30-5:30pm
1302 SC.

P-Fun Topics?

- Translation ER to Rel. model.
- RAT (10%)

  R.A. $\rightarrow$ SQL $\rightarrow$ $\boxed{\text{SQLite}}$ $\Rightarrow$ R.A. $\rightarrow$ $\boxed{\text{SQLite}}$

- Attr closure (today) - BCNF (tod.)
- F.D. closure (") - " " "

Functional Dependencies

- A form of constraint (hence, part of the schema)
- Finding them is part of the database design
- Used heavily in schema refinement

(SSN) Name Addr Phone

10 green
10 green

Name Address

Definition:

If two tuples agree on the attributes

T1 Alex ...
T2 Alex ...

Name $A_1, A_2, \ldots A_n$

then they must also agree on the attributes

Address $B_1, B_2, \ldots B_m$

Formally: $A_1, A_2, \ldots A_n \longrightarrow B_1, B_2, \ldots B_m$

Examples

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E1847	John	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

- EmpID $\longrightarrow$ Name, Phone, Position
- ✓ Position $\longrightarrow$ Phone
- but $\underbrace{\text{Phone}}_{1234}$ $\not\longrightarrow$ Position

clerk, Lawyer

In General

- To check if $A \longrightarrow B$ violation:

Erase all other columns

phone ← position

…	A	…	B	
	X1		Y1	
	X2		Y2	
	…		…	

one ← many / one

- check if the remaining relation is many-one (called ***functional*** in mathematics)

Example

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E1847	John	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	lawyer

No violation for Pos → Phone

More examples:

Product: name → price, manufacturer
Person: ssn → name, age
Company: name → stock price, president

Q: From this, can you conclude phone → SSN?

a phone is only used by ONE person.

SSN		Phone Number
123-321-99	Alex	(201) 555-1234
123-321-99	Alex	(206) 572-4312
909-438-44		(908) 464-0028
909-438-44		(212) 555-4000
123-321-88	Alex Junior	(201) 555-1234

F.D. stated at schema design

⇒ assertion

Relation Keys

NetID	Name	Addr	dept	age	...

- After defining FDs, we can now define keys
- Key of a relation R is a set of attributes that
  - functionally determines all attributes of R
  - none of its subsets determines all attributes of R
- Superkey
  - a set of attributes that contains a key
- We will need to know the keys of the relations in a DB schema, so that we can refine the schema

{N, A} → {NetID, dept, age, } ...

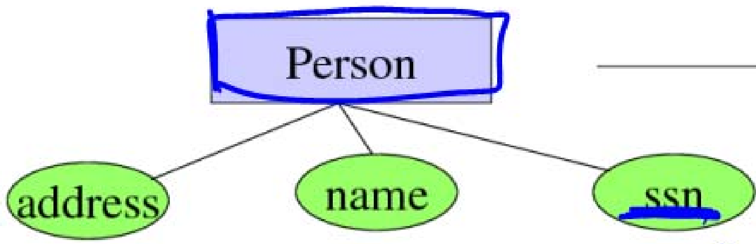
	(Name, Addr)	(NetID)	(NetID, dept)
key	✓	✓	✗
s.key	✓	✓	✓

Finding the Keys of a Relation

Given a relation constructed from an E/R diagram, what is its key?

Rules:

1. If the relation comes from an entity set,
   the key of the relation is the set of attributes which is the key of the entity set.

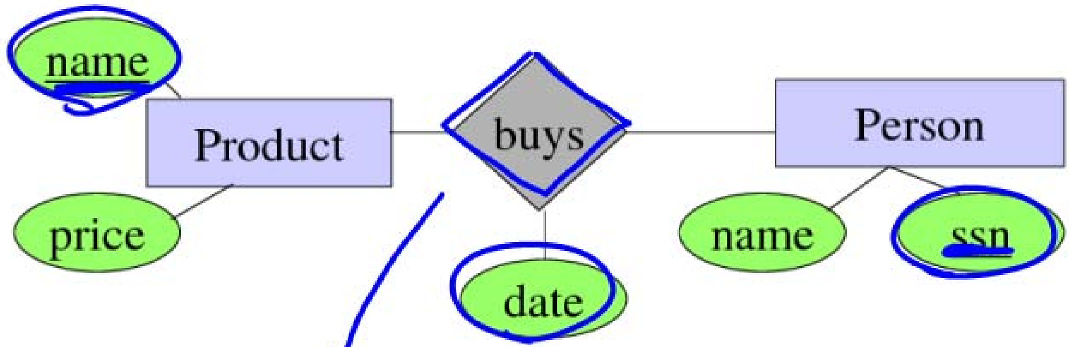


Person(address, name, ssn)

Finding the Keys

Rules:

2. If the relation comes from a many-many relationship, the key of the relation include the set of all attribute keys in the relations corresponding to the entity sets (and additional attributes if necessary)

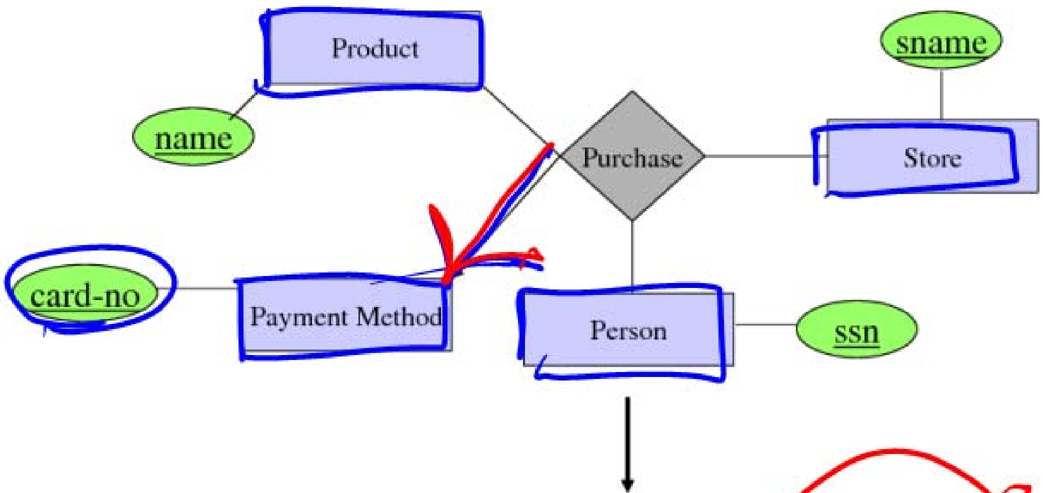


buys(name, ssn, date)

prod person

Finding the Keys

But: if there is an arrow from the relationship to E, then we don't need the key of E as part of the relation key.



Purchase(name , sname, ssn, card-no)

Finding the Keys

More specific rules:

- Many-one, one-many, one-one relationships
- Multi-way relationships
- Weak entity sets

(Try to find them yourself)

Reasoning with FDs

1) closure of FD sets
2) closure of attribute sets

You

Closure of FD sets

who gives FDs ?

- Given a relation schema R & a set S of FDs
  - is the FD f logically implied by S?

Name → age
age → drikable

Name
↓
drinkable

- Example
  - R = {A,B,C,G,H,I} 6 attrs.
  - S = A ➔B, A ➔C, CG ➔ H, CG ➔ I, B ➔ H
  - would A ➔ H be logically implied? new rule.
  - yes (you can prove this, using the definition of FD)
- Closure of S: S+ = $\boxed{\text{all FDs}}$ logically implied by S

such as A → H.

- How to compute S+?
  - we can use $\boxed{\text{Armstrong's}}$ axioms

Armstrong's Axioms Rules

proven by def.

- Reflexivity rule NetID → dept. addr
  - A1A2...An ➔ a subset of A1A2...An ⇒ NetID → dept
- Augmentation rule Name. NetID → dept. addr. Name
  - A1A2...An ➔ B1B2...Bm, then
    A1A2...An C1C2..Ck ➔ B1B2...Bm C1C2...Ck
- Transitivity rule
  - A1A2...An ➔ B1B2...Bm and
    same B1B2...Bm ➔ C1C2...Ck, then
    A1A2...An ➔ C1C2...Ck

Name → age
age → drink
⇒ Name → drink

Inferring S+ using Armstrong's Axioms

- S+ = S
- Loop
  - foreach f in S, apply reflexivity and augment. rules
  - add the new FDs to S+
  - foreach pair of FDs in S, apply the transitivity rule
  - add the new FD to S+
- Until S+ does not change any further

$S = \{ A \xrightarrow{f1} B,\ B \xrightarrow{f2} C,\ AC \xrightarrow{f3} D \}$

S^+ ?

"form changing"

so that $B \Rightarrow B$

① f_1, f_2. Tr $\Rightarrow$ + $A \to C$.

② (want to use $AC \to D$)

f2: $AB \to AC$

f1: $AA \to AB$

$\Rightarrow AA \to D \Rightarrow A \to D$

Name ______________ NetID ______________

Q1: What do you like best of this cls. that we must keep?

Q2: What .... dislike .... go?

Additional Rules

(aug)

$XX \to XY$

$XY \to YZ$ $\Rightarrow XX \to YZ$

- Union rule
  - $X \to Y$ and $X \to Z$, then $\boxed{X \to YZ}$

    netid dept netid addr netid dept addr
  - (X, Y, Z are sets of attributes)
- Decomposition rule
  - $X \to YZ$, then $X \to Y$ and $X \to Z$
- Pseudo-transitivity rule
  - $X \to Y$ and $YZ \to U$, then $XZ \to U$
- These rules can be inferred from Armstrong's axioms

Closure of a Set of Attributes

(name, addr) → ?

Given a set of attributes {***A1, …, An***} and a set of dependencies S.
Problem: find all attributes *B* such that:

any relation which satisfies S also satisfies:
A1, …, An → *B*

(You)

The **closure** of *{A1, …, An}*, denoted $\{A1, \ldots, An\}^+$,
is the set of all such attributes *B*

We will discuss the motivations for attribute closures soon ?

Is {name, addr} a key?

$\boxed{\{name, addr\}^+} \Rightarrow$ all attr.

Algorithm to Compute Closure

Start with X={A1, …, An}. {name, addr}

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

if $B_1, B_2, \ldots B_n \longrightarrow C$ is in S, **and**

$B_1, B_2, \ldots B_n$ are all in X, **and**

C is not in X

then

add C to X. X = X + {c}

Example

R: ⟨A, B, C, D, E, F⟩   Is (A, F) a key?

A B	→	C
A D	→	E
B	→	D
A F	→	B

Name Addr → age

✓ $(A,F)^+ = \{A, \ldots, F\}$

✗ $(A)^+ = \{\ldots\}$

✗ $(F)^+ = \{\ldots\}$

Closure of {A,B}: X = {A, B, C, D, E}

Closure of {A, F}: X = {A, F, B, D, C, E}

	(A, F) (A)?
AF→B	A, B, F
B→D	A, B, D, F
AD→E	A B D E F
AB→C	A B C D E F @stop

Usage for Attribute Closure

- Test if X is a superkey
  - compute X+, and check if X+ contains all attrs of R

- Check if X ➔ Y holds
  - by checking if Y is contained in X+

$Y \subseteq X^+ \iff X \rightarrow Y$

$$ER \dashrightarrow \underset{\text{original}}{\underline{\underline{R}}} \xrightarrow{?} \underset{\text{Normal form}}{\underline{\underline{R^*}}}$$

Desirable Properties of Schema Refinement

1) minimize redundancy
2) avoid info loss
3) preserve dependency
4) ensure good query performance

Normal Forms

x set. strig.

x array, Float.

First Normal Form = all attributes are atomic

Second Normal Form (2NF) = old and obsolete

SQL. Ted Codd.

Boyce Codd Normal Form (BCNF)

Third Normal Form (3NF)

Fourth Normal Form (4NF)

Others...

Boyce-Codd Normal Form

BCNF ? NO

SSN	addr	phone
Alex	10 G	123
Alex	10 G	456

A simple condition for removing anomalies from relations:

A relation R is in BCNF if and only if:

SSN ~~→~~ addr Bad

Whenever there is a nontrivial FD $A_1, A_2, \ldots A_n \longrightarrow B$ for R , it is the case that $\{ A_1, A_2, \ldots A_n \}$ is a super-key for R.

then

SSN is superkey

In English (though a bit vague):

X ?

SSN → ~~all attr~~

SSN, addr, phone

Whenever a set of attributes of *R* is determining another attribute, it should determine ***all*** attributes of *R*.

In Contrast

✓ SSN → addr
✓ SSN a key

yes

BCNF ?

SSN	addr
~~Alex~~	~~10 G~~
~~Alex~~	~~10 G~~

SSN	phone
Alex	123
Alex	456

Example

F.D. violate BCNF

Name	SSN	Phone Number
Fred	123-321-99	P1 (201) 555-1234
Fred	123-321-99	P2 (206) 572-4312
Joe	909-438-44	(908) 464-0028
Joe	909-438-44	(212) 555-4000

What are the dependencies?

SSN → Name

What are the keys?

Is it in BCNF?

Did u remove the F.D.?

Decompose it into BCNF

Not BCNF

$A \rightarrow B$

(addr)

SSN	Name	phone
A	B	C

T1 key

BCNF ✓

SSN	Name
123-321-99	Fred
909-438-44	Joe

SSN $\longrightarrow$ Name

C A B

T2

BCNF ?

SSN	Phone Number
123-321-99	(201) 555-1234
123-321-99	(206) 572-4312
909-438-44	(908) 464-0028
909-438-44	(212) 555-4000

A C

key = (SSN, phone)

What About This?

BCNF ? Yes

Name	Price	Category
Gizmo	$19.99	gadgets
OneClick	$24.99	camera

Name $\longrightarrow$ Price, Category

key

BCNF Decomposition

Find a dependency that violates the BCNF condition:

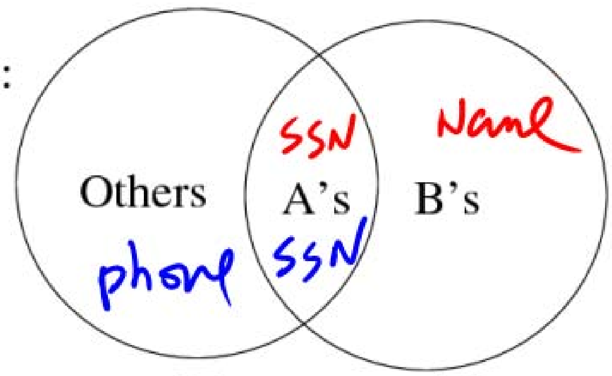
SSN Name

$$\underline{A_1, A_2, \ldots A_n} \longrightarrow \underline{B_1, B_2, \ldots B_m}$$

phone
C

Heuristics: choose B_1, B_2, … B_m "as large as possible"

Decompose:



R1 *R2*

Continue until there are no BCNF violations left.

Example Decomposition

SSN → Age → D.F.W

F.D. ✓

Person:

P

Name	SSN	Age	EyeColor	PhoneNumber	DraftWorthy

Functional dependencies:

✓ ① F.D. 1 SSN ⟶ (Name, Age, Eye Color) phone.

D.F.W

A B C

BNCF: P1 Person1(SSN, Name, Age, EyeColor), D.F.W A + B+

P2 Person2(SSN, PhoneNumber) A + C.

What if we also had an attribute Draft-worthy, and the FD:

② F.D. 2 Age ⟶ Draft-worthy

P11 (Age, d.w)

P12 (SSN, . . . , age)

BCNF Decomposition: The Algorithm

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

FD1 SSN → . . .
FD2 age → . . .

1) Compute S+
2) Compute keys for R (from ER or from S+)
3) Use S+ and keys to check if R is in BCNF, if not:
   - a) pick a violation FD f: A ➔ B
     one
     SSN → age, name, ec,
   - b) expand B as much as possible, by computing A+
     SSN → age, . . .
     DF.W
   - c) create R1 = A union B, R2 = A union (others in R)
   - d) compute all FDs over R1, using R and S+, then compute keys for R1. Repeat similarly for R2
   - e) Repeat Step 3 for R1 and R2
4) Stop when all relations are BCNF, or are two-attributes

Q: Is BCNF unique? NO

	SSN	NetID	phone
v	111	Alex1	123
o	222	Barb1	456
v	111	Alex1	321

F.D.1 SSN (A) → NetID (B) (c) phone

F.D.2 NetID → SSN

① F.D.1

SSN	NetID

SSN	phone

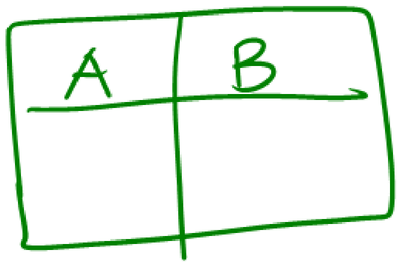
② F.D.2

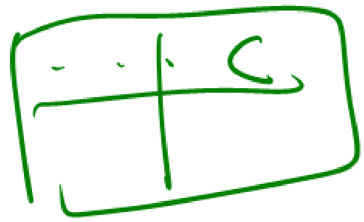
NetID	SSN

N.I.	phone

Q: Does BCNF always exist? Y

All two-attr tables are in BCNF.





$A \to B$. A must be key

$B \to A$, B must be key

Properties of BCNF

- BCNF removes certain types of redundancy
  - those caused by adding many-many or one-many relations
- For examples of redundancy that it cannot remove, see "multivalued redundancy"
- BCNF avoids information loss

ssn	addr	phone
111		123
111		321

many

Enrollment

many

name	dept	course	instr
Alex	CS	CS412	Johnson
Ben	EE	CS412	Johnson
Alex	CS	CS423	Many

Lossless Decompositions

A decomposition is *lossless* if we can recover:

R(A,B,C)

Decompose

{ R1(A,B) , R2(A,C) }

Recover

R'(A,B,C) = R(A,B,C)

R' is in general larger than R. Must ensure R' = R

Decomposition Based on BCNF is Necessarily Lossless

if (Alex, Mb, 'bud light')

name bar favorite

R(A, B, C), A → C

Name favorite

$C = c'$

$b \neq b'$

BCNF: R1(A,B), R2(A,C)

R1(Name, bar) R2(Name, Fav)

Some tuple (a,b,c) in R (Alex, J, bud) (a,b',c') also in R (Alex, G, bud)

decomposes into (a,b) in R1 (a,b') also in R1

and (a,c) in R2 (a,c') also in R2

No extra tuples

Recover tuples in R: (a,b,c), (a,b,c'), (a,b',c), (a,b',c') also in R ?

Can (a,b,c') be a bogus tuple? What about (a,b',c') ?

However,

- BCNF is not always dependency preserving
- In fact, some times we cannot find a BCNF decomposition that is dependency preserving
- Can handle this situation using 3NF
- See next few slides for example

Ted Codd : R. m. 1970

Behind the Scene: The Great Debate of '75

- The network/COBOL camp:
  - DBTG (Database Task Group, under CODASYL), 1971
  - closely aligned with COBOL
  - DBTG Report would standardize network model
  - Bachman (for network model) got Turing award in 1973
- The relational camp:
  - Codd's paper in 1970
  - resistance even within IBM
  - First implementations, 1973: System R (IBM), INGRES (Berkeley)
  - System R at IBM San Jose Lab
- The "Great Debate" in 1975 SIGMOD conf.
- Codd got Turing award in 1981

Behind the Scene: Arguments Against the Other Side?

Network

- COBOL/CODASYL → Relational
  - too mathematical (to understand)

Network

- Relational → COBOL/CODASYL
  - too complicated (to program)

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...

3NF: A Problem with BCNF

Unit	Company	Product

FD's: Unit → Company; Company, Product → Unit

So, there is a BCNF violation, and we decompose.

Unit	Company

Unit → Company

Unit	Product

No FDs

So What's the Problem?

Unit	Company
Galaga99	UI
Bingo	UI

Unit	Product
Galaga99	databases
Bingo	databases

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

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

Unit	Company	Product
Galaga99	UI	databases
Bingo	UI	databases

Violates the dependency: company, product -> unit!

Preserving FDs

- What if, when a relation is decomposed, the X of an $X \rightarrow Y$ ends up only in one of the new relations and the Y ends up only in another?
- Such a decomposition is not "dependency-preserving."
- Goal: Always have FD-preserving decompositions

Solution: 3rd Normal Form (3NF)

A simple condition for removing anomalies from relations:

A relation R is in 3rd normal form if :

Whenever there is a nontrivial dependency $A_1, A_2, \ldots, A_n \rightarrow B$ for R , then $\{A_1, A_2, \ldots, A_n\}$ is a super-key for R,
or B is part of a key.

3NF (General Definition)

- A relation is in Third Normal Form (3NF) if whenever X→A holds, either X is a superkey, or A is a prime attribute.

Informally: everything depends on the key or is in a key.

- Despite the thorny technical definitions that lead up to it, 3NF is intuitive and not hard to achieve. ***Aim for it in all designs unless you have strong reasons otherwise.***

3NF vs. BCNF

- R is in BCNF if whenever $X \rightarrow A$ holds, then X is a superkey.
- Slightly stronger than 3NF.
- Example: $R(A,B,C)$ with $\{A,B\} \rightarrow C$, $C \rightarrow A$
  - 3NF but not BCNF

Guideline: Aim for BCNF and settle for 3NF

Decomposing R into 3NF

- The algorithm is complicated
- 1. Get a "minimal cover" of FDs
- 2. Find a lossless-join decomposition of R (which might miss dependencies)
- 3. Add additional relations to the decomposition to cover any missing FDs of the cover
- Result will be lossless, will be dependency-preserving 3NF; might not be BCNF

- This way equivalent to textbook, but easier to follow.
- → *Example 3.27 in textbook.*

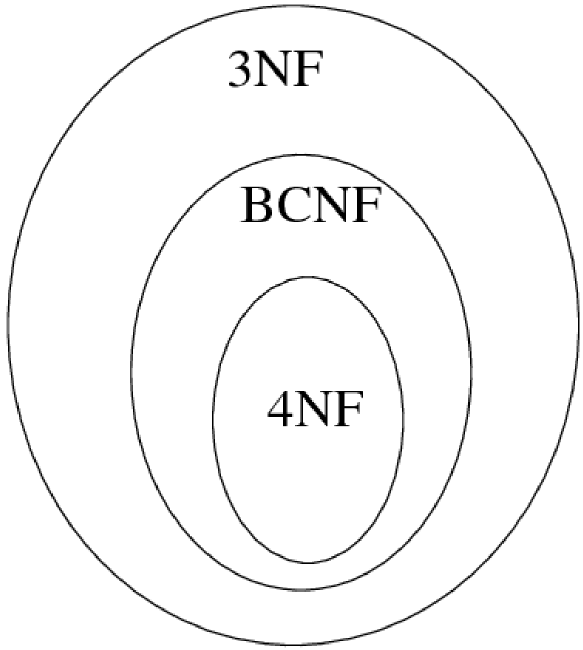
Fact of life...

Finding a decomposition which is both lossless and dependency-preserving is not always possible.

Multi-valued Dependencies and 4NF

we will not cover this.

Confused by Normal Forms ?



In practice: (1) 3NF is enough, (2) don't overdo it !

Normalization Summary

- 1NF: usually part of the woodwork
- 2NF: usually skipped
- 3NF: a biggie
  - always aim for this
- BCNF and 4NF: tradeoffs start here
  - in re: d-preserving and losslessness
- 5NF: You can say you've heard of it...

Caveat

- Normalization is not the be-all and end-all of DB design
- Example: suppose attributes A and B are always used together, but normalization theory says they should be in different tables.
  – decomposition might produce unacceptable performance loss (extra disk reads)
- Plus -- there are constraints other than FDs and MVDs

Current Trends

Normalization ≠ crucial

- Object DBs and Object-Relational DB's
  - may permit complex attributes
  - 1st normal form unnecessary
- Data Warehouses
  - huge historical databases, seldom or never updated after creation
  - joins expensive or impractical
  - argues against normalization
- Everyday relational DBs
  - aim for BCNF, settle for 3NF