

# Normalization is a Good Idea

Eugene Wu

Fall 2018

# Steps for a New Application

## Requirements

what are you going to build?

## Conceptual Database Design

pen-and-pencil description

## Logical Design

formal database schema

## Schema Refinement:

fix potential problems, normalization

Normalization

## Physical Database Design

use sample of queries to optimize for speed/storage

## App/Security Design

prevent security problems

# A Relational Model of Data for Large Shared Data Banks

E. F. Codd

*IBM Research Laboratory, San Jose, California*

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report

# Redundancy = Bad

Update/insert/delete anomalies. Wastes space

<u>sid</u>	name	address	hobby	cost
1	Eugene	amsterdam	trucks	\$\$
1	Eugene	amsterdam	cheese	\$
2	Bob	40th	paint	\$\$\$
3	Bob	40th	cheese	\$
4	Shaq	florida	swimming	\$

people have names and addrs

hobbies have costs

people many-to-many with hobbies

What's primary key? sid? sid + hobby?

# Anomalies (Inconsistencies)

## Update Anomaly

change one address, need to change all

## Insert Anomaly

add person without hobby?  
not allowed? dummy hobby?

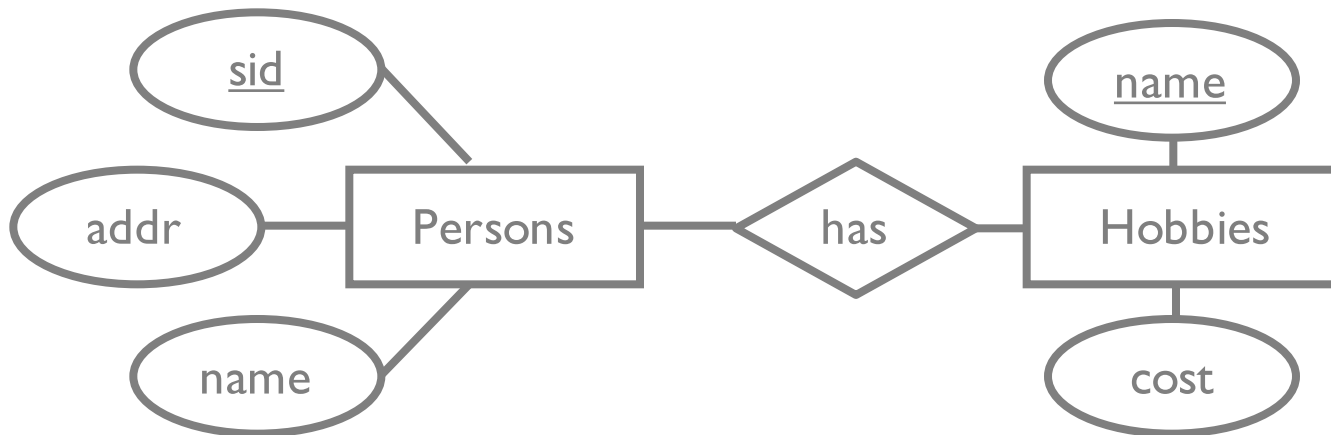
## Delete Anomaly

if delete a hobby. Delete the person?

**Theory Can Fix This!**

# A Possible Approach

ER diagram was a heuristic



We have decomposed example table into:

person(sid, addr, name)

hobby(name, cost)

personhobby(hobbyname, sid)

**WHY is this a good decomposition??**

# A Possible Approach

What if decompose into:

person(sid, name, address, cost)

personhobby(sid, hobbyname)

<u>sid</u>	name	address	cost
1	Eugene	amsterdam	\$\$
1	Eugene	amsterdam	\$
2	Bob	40th	\$\$\$
3	Bob	40th	\$
4	Shaq	florida	\$

<u>sid</u>	hobby
1	trucks
1	cheese
2	paint
3	cheese
4	swimming

but... which cost goes with which hobby?

lost information: *lossy decomposition*

# Decomposition

Replace schema R with 2+ smaller schemas that

1. each contain subset of attrs in R
2. together include all attrs in R

ABCD replaced with AB, BCD or AB, BC, CD

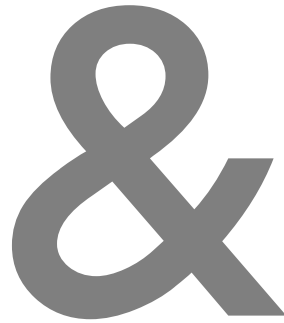
Not free – may introduce problems!

1. lossy-join: able to recover R from smaller relations
2. non-dependency-preserving: constraints on R hold by only enforcing constraints on smaller schemas
3. performance: additional joins, may affect performance



Can we systematically  
decompose our relation to

prevent  
decomposition  
problems



remove  
redundancy?

# Functional Dependencies (FD)

sid	name	address	hobby	cost
1	Eugene	amsterdam	trucks	\$\$
1	Eugene	amsterdam	cheese	\$
2	Bob	40th	paint	\$\$\$
3	Bob	40th	cheese	\$
4	Shaq	florida	swimming	\$

sid sufficient to identify name and addr, but not hobby  
e.g., exists a function  $f(\text{sid}) \rightarrow \text{name, addr}$

sid  $\rightarrow$  name, addr is a **functional dependency**

“sid determines name, addr”

“name, addr are functionally dependent on sid”

“if 2 records have the same sid, their name and addr are the same”

# Functional Dependencies (FD)

$$X \rightarrow Y$$

holds on R

if  $t_1.X = t_2.X$  then  $t_1.Y = t_2.Y$

where  $X, Y$  are subsets of attrs in R

## Examples of FDs in person-hobbies table

$\text{sid, hobby} \rightarrow \text{name, address cost}$

$\text{hobby} \rightarrow \text{cost}$

$\text{sid} \rightarrow \text{name, address}$

# Fun Facts

Functional Dependency is an integrity constraint  
statement about all instances of relation

Generalizes key constraints

if  $K$  is candidate key of  $R$ , then  $K \rightarrow R$

Given FDs, simple definition of redundancy

when left side of FD is not table key

Where do FDs come from?

thinking really hard aka application semantics

can't stare at database to derive (like ICs)

# Fun Facts

Functional Dependency is an integrity constraint  
statement about all instances of relation

Generalizes key constraints

if  $K$  is candidate key of  $R$ , then  $K \rightarrow R$

## **Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms**

Thorsten Papenbrock<sup>2</sup>

Jens Ehrlich<sup>1</sup>

Jannik Marten<sup>1</sup>

Tommy Neubert<sup>1</sup>

Jan-Peer Rudolph<sup>1</sup>

Martin Schönberg<sup>1</sup>

Jakob Zwiener<sup>1</sup>

Felix Naumann<sup>2</sup>

<sup>1</sup> firstname.lastname@student.hpi.uni-potsdam.de

<sup>2</sup> firstname.lastname@hpi.de

Hasso-Plattner-Institut, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany

# Normal Forms

Two different criterias for decomposing a relation

Boyce Codd Normal Form (BCNF)

No redundancy, may lose dependencies

Third Normal Form (3NF)

May have redundancy, no decomposition problems

## Redundancy depends on FDs

consider  $R(ABC)$

no FDs: no redundancy

if  $A \rightarrow B$ : tuples with same A value means B is duplicated!



# BCNF

Relation R in BCNF has *no redundancy* wrt FDs  
(only FDs are key constraints)

F: set of functional dependencies over relation R

for  $(X \rightarrow Y)$  in F  
Y is in X *or*  
X is a superkey of R

Is this in BCNF?

$\text{sid} \rightarrow \text{name}$

sid	hobby	name
x	$y_1$	z
x	$y_2$	?



# BCNF

Relation R in BCNF has *no redundancy* wrt FDs  
(only FDs are key constraints)

F: set of functional dependencies over relation R

for  $(X \rightarrow Y)$  in F  
Y is in X *or*  
X is a superkey of R

Functional Dependencies

$SH \rightarrow NAC$  (sid, hobby  $\rightarrow$  name, addr, cost)

$H \rightarrow C$

$S \rightarrow NA$

What's in BCNF?

SHNAC NO

SNA, SHC NO

SNA, HC, SH YES

# BCNF

Suppose we have

Client, Office  $\rightarrow$  Account

Account  $\rightarrow$  Office

What's in BCNF?

R(Account, Client, Office)

R(Account, Office)   R(Client, Account)

Where did  $CO \rightarrow A$  go? *Lost a Functional Dependency*

Can we preserve FDs *and* remove most redundancy?

# 3<sup>rd</sup> Normal Form (3NF)

Relax BCNF (e.g.,  $BCNF \subseteq 3NF$ )

F: set of functional dependencies over relation R

for  $(X \rightarrow Y)$  in F

Y is in X *or*

X is a superkey of R *or*

# 3<sup>rd</sup> Normal Form (3NF)

Relax BCNF (e.g.,  $BCNF \subseteq 3NF$ )

F: set of functional dependencies over relation R

for  $(X \rightarrow Y)$  in F

Y is in X *or*

X is a superkey of R *or*

Y is part of a key in R

Is new condition trivial? NO! key is minimal

Nice properties

lossless join ^ dependency preserving decomposition to 3NF always possible

# 3<sup>rd</sup> Normal Form (3NF)

Relax BCNF (e.g.,  $BCNF \subseteq 3NF$ )

F: set of functional dependencies over relation R

for  $(X \rightarrow Y)$  in F

Y is in X *or*

X is a superkey of R *or*

Y is part of a key in R

FDs

$CO \rightarrow A$

$A \rightarrow O$

(AO), (CA) splits up key in  $CO \rightarrow A$   
COA is in 3NF!

# 3<sup>rd</sup> Normal Form (3NF)

Relax BCNF (e.g.,  $BCNF \subseteq 3NF$ )

F: set of functional dependencies over relation R

for  $(X \rightarrow Y)$  in F

Y is in X *or*

X is a superkey of R *or*

Y is part of a key in R

Schema: SBDC

$SBD \rightarrow C, S \rightarrow C$

Not in 3NF

$SBD \rightarrow C, S \rightarrow C, C \rightarrow S$

In 3NF (Hint: CBD is a key)

In both cases, SC is stored redundantly

# We're going to need some theory

Closure of FDs

armstrong's axioms

Minimal FD Set

Principled Decomposition

BCNF & 3NF

# Closure of FDs

If I know

$\text{Name} \rightarrow \text{Bday}$  and  $\text{Bday} \rightarrow \text{age}$

Then it implies

$\text{Name} \rightarrow \text{age}$

$f'$  is implied by set  $F$  if  $f'$  is true when  $F$  is true

$F^+$  **closure** of  $F$  is all FDs implied by  $F$

Can we construct this closure automatically? YES



# Closure of FDs

*Inference rules* called **Armstrong's Axioms**

Reflexivity      if  $Y \subseteq X$  then  $X \rightarrow Y$

Augmentation    if  $X \rightarrow Y$  then  $XZ \rightarrow YZ$  for any  $Z$

Transitivity      if  $X \rightarrow Y$  &  $Y \rightarrow Z$  then  $X \rightarrow Z$

These are **sound** and **complete** rules

sound            doesn't produce FDs not in the closure

complete        doesn't miss any FDs in the closure

# Closure of FDs

Can we compute the closure?    YES. slowly  
expensive. exponential in # attributes

Can we *check* if  $X \rightarrow Y$  is in the closure of F?

$X^+ = \text{attribute closure}$  of X (expand X using axioms)

check if Y is implied in the attribute closure

# Closure of FDs

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

Is  $A \rightarrow E$  in the closure?

$A \rightarrow B$

given

$A \rightarrow AB$

augmentation  $A$

$A \rightarrow BB$

apply  $A \rightarrow B$  (transitivity)

$A \rightarrow BC$

apply  $B \rightarrow C$  (transitivity)

$A \rightarrow E$

apply  $BC \rightarrow E$  (transitivity)

# We're going to need some theory

Closure of FDs

armstrong's axioms

Minimal FD Set

Principled Decomposition

BCNF & 3NF

# Minimum Cover of FDs

Closures let us compare sets of FDs meaningfully

$$F1 = \{A \rightarrow B, A \rightarrow C, A \rightarrow BC\}$$

$$F2 = \{A \rightarrow B, A \rightarrow C\}$$

F1 equivalent to F2

If there's a closure (a maximally expanded FD), there's a *minimal* FD. Let's find it

# Minimum Cover of FDs

1. Turn FDs into *standard form*  
decompose each FD so single attr on the right side
2. Minimize left side of each FD  
for each FD, check if can delete left attr w/out changing closure  
given  $ABC \rightarrow D, B \rightarrow C$  can reduce to  $AB \rightarrow D, B \rightarrow C$
3. Delete redundant FDs  
check each remaining FD and see if it can be deleted  
e.g., in closure of the other FDs

2 must happen before 3!

# Minimum Cover of FDs

$A \rightarrow B, ABC \rightarrow E, EF \rightarrow G, ACF \rightarrow EG$

Standard form

$A \rightarrow B, ABC \rightarrow E, EF \rightarrow G, ACF \rightarrow E, ACF \rightarrow G$

Minimize left side

$A \rightarrow B, AC \rightarrow E, EF \rightarrow G, ACF \rightarrow E, ACF \rightarrow G$

reason:  $AC \rightarrow E + A \rightarrow B$  implies  $ABC \rightarrow E$

Delete Redundant FDs

$A \rightarrow B, AC \rightarrow E, EF \rightarrow G, \cancel{ACF \rightarrow E}, \cancel{ACF \rightarrow G}$

reason:  $ACF \rightarrow E$  implied by  $AC \rightarrow E, EF \rightarrow G$

# We're going to need some theory

Closure of FDs

armstrong's axioms

Minimal FD Set

Principled Decomposition

BCNF & 3NF



# Decomposition

Eventually want to decompose  $R$  into  $R_1 \dots R_n$  wrt  $F$

We've seen issues with decomposition.

Lost Joins: Can't recover  $R$  from  $R_1 \dots R_n$

Lost dependencies

Principled way of avoiding these?

# Lossless Join Decomposition

Let's say relation  $R$  is decomposed into relations  $X, Y$

Join the decomposed tables to get *exactly the* original

$$\pi_X(R) \bowtie \pi_Y(R) = R$$

Lossless wrt  $F$  if and only if  $F^+$  contains

$$X \cap Y \rightarrow X \text{ or } Y \cap X \rightarrow Y$$

intersection of  $X, Y$  is a key for one of them

# Lossless Join Decomposition

Lossless wrt  $F$  if and only if  $F^+$  contains

$$X \cap Y \rightarrow X \text{ or } Y \cap X \rightarrow Y$$

intersection of  $X, Y$  is a key for one of them

FDs:  $A \rightarrow C, A \rightarrow B$

A	B	C
1	2	1
5	3	4
9	2	6



A	B
1	2
5	3
9	2

B	C
2	1
3	4
2	6



A	B	C
1	2	1
5	3	4
9	2	6
1	2	6
9	2	1

Lossy!  $AB \cap BC = B$  doesn't determine anything

# Lossless Join Decomposition


Lossless wrt  $F$  if and only if  $F^+$  contains

$$X \cap Y \rightarrow X \text{ or } Y \cap X \rightarrow Y$$

intersection of  $X, Y$  is a key for one of them


FDs:  $A \rightarrow C, A \rightarrow B$

A	B	C
1	2	1
5	3	4
9	2	6



A	B
1	2
5	3
9	2

A	C
1	1
5	4
9	6



A	B	C
1	2	1
5	3	4
9	2	6

OK

# Dependency-preserving Decomposition

$F_R$  = Projection of  $F$  onto  $R$

Subset of  $F$  that are “valid” for  $R$

FDs  $X \rightarrow Y$  in  $F^+$  s.t.  $X$  and  $Y$  attrs are in  $R$

If  $R$  decompose to  $X, Y$ .

FDs that hold on  $X, Y$  equivalent to all FDs on  $R$

$$(F_X \cup F_Y)^+ = F^+$$

Consider  $ABCD$ ,  $C$  is key,  $AB \rightarrow C$ ,  $D \rightarrow A$

BCNF decomposition:  $BCD$ ,  $DA$

$AB \rightarrow C$  doesn't apply to either table!

# We're going to need some theory

Closure of FDs

armstrong's axioms

Minimal FD Set

Principled Decomposition

BCNF & 3NF

# BCNF

while BCNF is violated

  R with FDs  $F_R$

  if  $X \rightarrow Y$  violates BCNF

    turn R into R-Y & XY

ABCDE    key A,  $BC \rightarrow A$ ,  $D \rightarrow B$ ,  $C \rightarrow D$

DB, ACDE            using  $D \rightarrow B$

DB, CD, ACE        using  $C \rightarrow D$

uh oh, lost  $BC \rightarrow A$

# 3NF

$F^{\min}$  = minimal cover of  $F$

Run BCNF using  $F^{\min}$

for  $X \rightarrow Y$  in  $F^{\min}$  not in projection onto  $R_1 \dots R_N$   
create relation  $XY$

ABCDE    key A,  $BC \rightarrow A$ ,  $D \rightarrow B$ ,  $C \rightarrow D$



# 3NF

$F^{\min}$  = minimal cover of  $F$

Run BCNF using  $F^{\min}$

for  $X \rightarrow Y$  in  $F^{\min}$  not in projection onto  $R_1 \dots R_N$   
create relation  $XY$

ABCDE    key A,  $C \rightarrow A$ ,  $D \rightarrow B$ ,  $C \rightarrow D$

DB, ACDE

DB, CD, ACE

DB, CD, AC, CE

# Summary

Normal Forms: BCNF and 3NF

FD closures: Armstrong's axioms

Proper Decomposition

# Summary

Accidental redundancy is really really bad

Adding lots of joins can hurt performance

Can be at odds with each other

Normalization good starting point, relax as needed

People usually think in terms of entities and keys,  
usually ends up reasonable

# What you should know

Purpose of normalization

Anomalies

Decomposition problems

Functional dependencies & axioms

3NF & BCNF

properties

algorithm