CMSC424: Normalization

Relational Database Design

- Lots of different choices for a schema
  - How do we decide between them
- If from an E-R diagram, then:
  - Did we make the right decisions with the E-R diagram ?
- Goals:
  - Formal definition of what it means to be a “good” schema.
  - How to achieve it.

Movies Database Schema

Movie(*title, year*, length, inColor, studioName, producerID)

StarsIn(movieTitle, movieYear, starName)

MovieStar(*name*, address, gender, birthdate)

MovieExec(name, address, *ID,* netWorth)

Studio(*name*, address, lawyerID)

Changed to:

Movie(*title, year*, length, inColor, studioName, producerID, starName)

<StarsIn merged into above>

MovieStar(*name*, address, gender, birthdate)

MovieExec(name, address, *ID,* netWorth)

Studio(*name*, address, lawyerID)

Is this a good schema ???

Movie(*title, year*, length, inColor, studioName, producerID, starName)

Title	**Year**	**Length**	**inColor**	**StudioName**	**prodID**	**StarName**
Star wars	1977	121	Yes	Fox	128	Hamill
Star wars	1977	121	Yes	Fox	128	Fisher
Star wars	1977	121	Yes	Fox	128	H. Ford
King Kong	2005	187	Yes	Universal	150	Watts
King Kong	1933	100	no	RKO	20	Fay

Issues:

1. Redundancy ➔ higher storage, inconsistencies ("anomalies")

   update anomalies, insertion anamolies

2. Need nulls

   Unable to represent some information without using nulls

   How to store movies w/o actors (pre-productions etc) ?

Movie(*<u>title, year</u>*, length, inColor, studioName, producerID, <u>starNames</u>)

Title	Year	Length	inColor	StudioName	prodID	StarNames
Star wars	1977	121	Yes	Fox	128	{Hamill, Fisher, H. ford}
King Kong	2005	187	Yes	Universal	150	Watts
King Kong	1933	100	no	RKO	20	Fay

<u>Issues:</u>

3. Avoid sets

- Hard to represent
- Hard to query

Smaller schemas always good ????

Split Studio(*name*, address, lawyerID) into:

Studio1 (name, lawyerID)

Name	**lawyerID**
Fox	101
Studio2	101
Universial	102

Studio2(name, address)???

Name	**Address**
Fox	Address1
Studio2	Address1
Universial	Address2

This process is also called *"decomposition"*

Issues:

4. Requires more joins (w/o any obvious benefits)

5. Hard to check for some dependencies

What if the "address" is actually the lawyer's address ?

No easy way to ensure that constraint (w/o a join).

Smaller schemas always good ????

Decompose StarsIn(<u>movieTitle</u>, <u>movieYear, starName</u>) into:

StarsIn1(movieTitle, movieYear)

StarsIn2(movieTitle, starName) ???

movieTitle	movieYear
Star wars	1977
King Kong	1933
King Kong	2005

movieTitle	starName
Star Wars	Hamill
King Kong	Watts
King Kong	Faye

<u>Issues:</u>

6. "joining" them back results in more tuples than what we started with

(King Kong, 1933, Watts) & (King Kong, 2005, Faye)

This is a "lossy" decomposition

We lost some constraints/information

The previous example was a "lossless" decomposition.

What we want …

- No sets
- Correct and faithful to the original design
  - Avoid lossy decompositions
- As little redundancy as possible
  - To avoid potential anomalies
- No "inability to represent information"
  - Nulls shouldn't be required to store information
- Dependency preservation
  - Should be possible to check for constraints

Not always possible.

We sometimes relax these for:

simpler schemas, and *fewer joins during queries.*

Approach

1. We will encode and list all our knowledge about the schema

- Functional dependencies (FDs)

  SSN → name (means: SSN "implies" length)

- If two tuples have the same "SSN", they must have the same "name"

  movietitle → length ???? Not true.

- But, (movietitle, movieYear) → length --- True.

2. Define a set of rules that the schema must follow to be considered good

- "Normal forms": 1NF, 2NF, 3NF, BCNF, 4NF, …
- A normal form specifies constraints on the schemas and FDs

3. If not in a "normal form", we modify the schema

FDs: Example 1

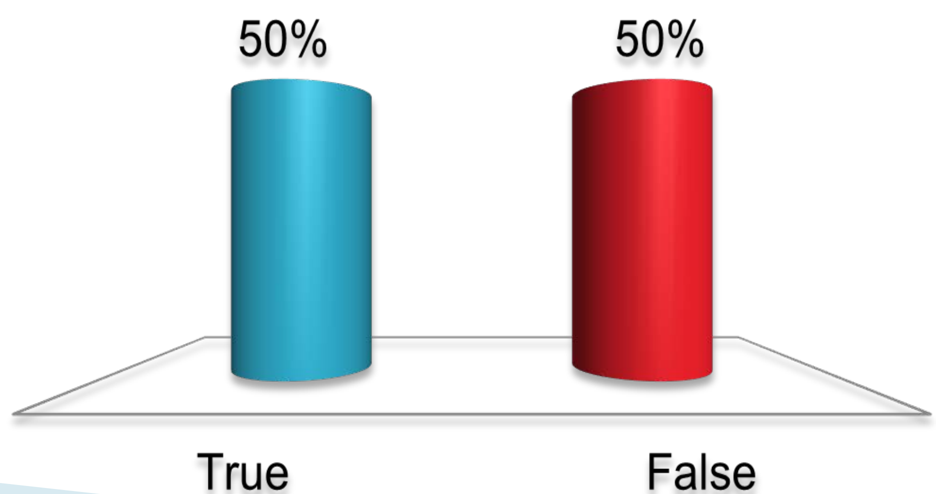
Title	Year	Length	StarName	Birthdate	producerID	Producer -address	Prodcuer -name	netWorth
Plane Crazy	1927	6	NULL	NULL	WD100	Mickey Rd	Walt Disney	100000
Star Wars	1977	121	H. Ford	7/13/42	GL102	Tatooine	George Lucas	10^9
Star Wars	1977	121	M. Hamill	9/25/51	GL102	Tatooine	George Lucas	10^9
Star Wars	1977	121	C. Fisher	10/21/56	GL102	Tatooine	George Lucas	10^9
King Kong	1933	100	F. Wray	9/15/07	MC100	NULL	NULL	NULL
King Kong	2005	187	N. Watts	9/28/68	PJ100	Middle Earth	Peter Jackson	10^8

ProducerID → Producer-address is a FD?

Title	Year	Length	StarName	Birthdate	producerID	Producer -address	Prodcuer -name	netWorth
Plane Crazy	1927	6	NULL	NULL	WD100	Mickey Rd	Walt Disney	100000
Star Wars	1977	121	H. Ford	7/13/42	GL102	Tatooine	George Lucas	10^9
Star Wars	1977	121	M. Hamill	9/25/51	GL102	Tatooine	George Lucas	10^9
Star Wars	1977	121	C. Fisher	10/21/56	GL102	Tatooine	George Lucas	10^9
King Kong	1933	100	F. Wray	9/15/07	MC100	NULL	NULL	NULL
King Kong	2005	187	N. Watts	9/28/68	PJ100	Middle Earth	Peter Jackson	10^8

A. True

B. False

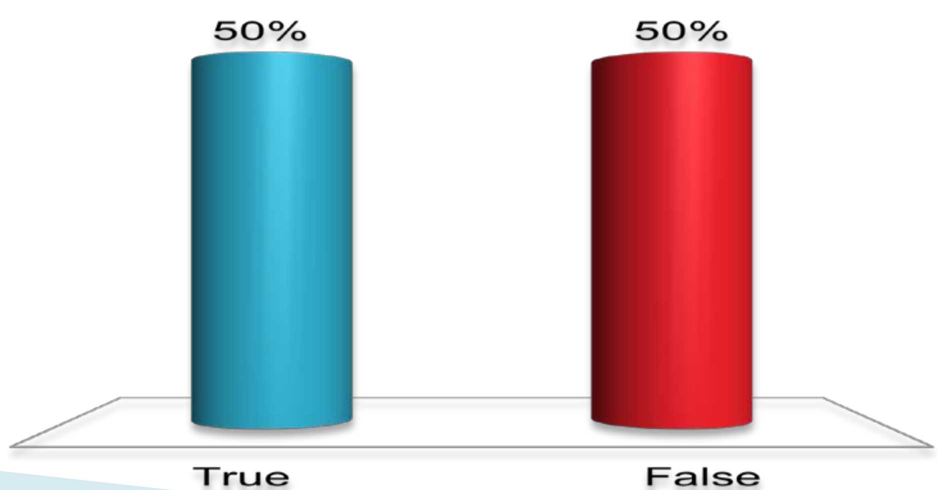


StarName → Birthdate is a FD?

Title	Year	Length	StarName	Birthdate	producerID	Producer -address	Prodcuer -name	netWorth
Plane Crazy	1927	6	NULL	NULL	WD100	Mickey Rd	Walt Disney	100000
Star Wars	1977	121	H. Ford	7/13/42	GL102	Tatooine	George Lucas	10^9
Star Wars	1977	121	M. Hamill	9/25/51	GL102	Tatooine	George Lucas	10^9
Star Wars	1977	121	C. Fisher	10/21/56	GL102	Tatooine	George Lucas	10^9
King Kong	1933	100	F. Wray	9/15/07	MC100	NULL	NULL	NULL
King Kong	2005	187	N. Watts	9/28/68	PJ100	Middle Earth	Peter Jackson	10^8

A. True

B. False

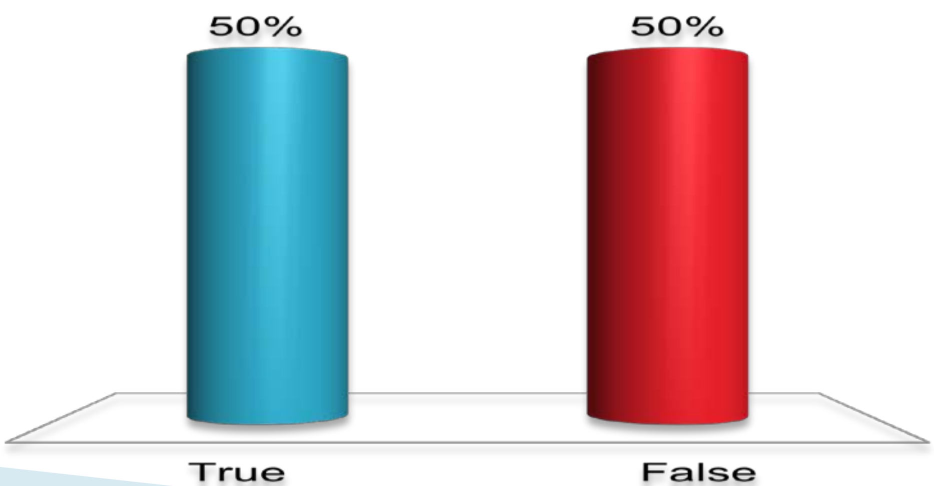


netWorth → Producer-name is a FD?

Title	Year	Length	StarName	Birthdate	producerID	Producer –address	Prodcuer –name	netWorth
Plane Crazy	1927	6	NULL	NULL	WD100	Mickey Rd	Walt Disney	100000
Star Wars	1977	121	H. Ford	7/13/42	GL102	Tatooine	George Lucas	10^9
Star Wars	1977	121	M. Hamill	9/25/51	GL102	Tatooine	George Lucas	10^9
Star Wars	1977	121	C. Fisher	10/21/56	GL102	Tatooine	George Lucas	10^9
King Kong	1933	100	F. Wray	9/15/07	MC100	NULL	NULL	NULL
King Kong	2005	187	N. Watts	9/28/68	PJ100	Middle Earth	Peter Jackson	10^9

A. True

B. False



Year → Length is a FD (for this instance)?

Title	Year	Length	StarName	Birthdate	producerID	Producer -address	Prodcuer -name	netWorth
Plane Crazy	1927	6	NULL	NULL	WD100	Mickey Rd	Walt Disney	100000
Star Wars	1977	121	H. Ford	7/13/42	GL102	Tatooine	George Lucas	10^9
Star Wars	1977	121	M. Hamill	9/25/51	GL102	Tatooine	George Lucas	10^9
Star Wars	1977	121	C. Fisher	10/21/56	GL102	Tatooine	George Lucas	10^9
King Kong	1933	100	F. Wray	9/15/07	MC100	NULL	NULL	NULL
King Kong	2005	187	N. Watts	9/28/68	PJ100	Middle Earth	Peter Jackson	10^9

A. True

B. False

Functional Dependencies

Difference between holding on an *instance* and holding on *all legal relation*

Title	Year	Length	inColor	StudioName	prodID	StarName
Star wars	1977	121	Yes	Fox	128	Hamill
Star wars	1977	121	Yes	Fox	128	Fisher
Star wars	1977	121	Yes	Fox	128	H. Ford
King Kong	1933	100	no	RKO	20	Fay

Year → Length *holds on this instance*

Is this a true functional dependency ? ***No****.*

Two movies in same year can have different lengths.

Can't draw conclusions based on a *single instance*

Need to use domain knowledge to decide which FDs hold

Functional Dependencies

- Functional dependencies and *keys*
  - A *key* constraint is a specific form of a FD.
  - E.g. if *A* is a superkey for *R*, then:

    $$A \rightarrow R$$

  - Similarly for *candidate keys and primary keys.*

- Deriving FDs
  - A set of FDs may imply other FDs
  - *e.g. If $A \rightarrow B$, and $B \rightarrow C$, then clearly $A \rightarrow C$*
  - *Book contains a formal method for inferring this (not in assigned reading)*

Definitions

1. A relation instance *r* *satisfies* a set of functional dependencies, *F*, if the FDs hold on the relation

2. F holds on a relation schema *R* if no legal (allowable) relation instance of *R* violates it

3. A functional dependency, *A → B,* is called *trivial* if:
   - B is a subset of A
   - e.g. Movieyear, length → length

Approach

1. We will encode and list all our knowledge about the schema
   - Functional dependencies (FDs)
   - Also:
     - Multi-valued dependencies (briefly discuss later)
     - Join dependencies etc…
2. Define a set of rules that the schema must follow to be considered good
   - "Normal forms": 1NF, 2NF, 3NF, BCNF, 4NF, …
   - A normal form specifies constraints on the schemas and FDs
3. If not in a "normal form", we modify the schema

BCNF: Boyce-Codd Normal Form

- A relation schema *R* is "in BCNF" if:
  - Every functional dependency $A \rightarrow B$ that holds on it is *EITHER*:
    1. Trivial *OR*
    2. *A* is a *superkey* of *R*

- *Why is BCNF good ?*
  - Guarantees that there can be no redundancy because of a functional dependency
  - Consider a relation *r(A, B, C, D)* with functional dependency $A \rightarrow B$ and two tuples: (a1, b1, c1, d1), and (a1, b1, c2, d2)
    - *b1* is repeated because of the functional dependency
    - BUT this relation is not in BCNF
      - $A \rightarrow B$ is neither trivial nor is *A* a superkey for the relation

BCNF and Redundancy

- *Why does redundancy arise ?*
  - Given a FD, A → B, if A is repeated (B – A) has to be repeated
  1. If rule 1 is satisfied, (B – A) is empty, so not a problem.
  2. If rule 2 is satisfied, then A can't be repeated, so this doesn't happen either

- Hence no redundancy because of FDs
  - Redundancy may exist because of other types of dependencies
    - Higher normal forms used for that (specifically, 4NF)
  - Data may naturally have duplicated/redundant data
    - We can't control that unless a FD or some other dependency is defined

Approach

1. We will encode and list all our knowledge about the schema
   - Functional dependencies (FDs); Multi-valued dependencies; Join dependencies etc…
2. We will define a set of rules that the schema must follow to be considered good
   - “Normal forms”: 1NF, 2NF, 3NF, BCNF, 4NF, …
   - A normal form specifies constraints on the schemas and FDs
3. If not in a “normal form”, we modify the schema
   - Through lossless decomposition (splitting)
   - Or direct construction using the dependencies information

BCNF

- Given a relation schema *R,* and a set of functional dependencies *F,* if every FD, *A* $\rightarrow$ *B,* is either:
  1. Trivial
  2. *A* is a *superkey* of *R*

  Then, *R* is in *BCNF (Boyce-Codd Normal Form)*

- What if the schema is not in BCNF ?
  - *Decompose (split) the schema into two pieces.*
  - Careful: you want the decomposition to be lossless

Achieving BCNF Schemas

For all dependencies *A* → *B* that hold on a relation,

check if *A* is a superkey

If not, then

Choose a dependency that breaks the BCNF rules, say A → B

Create R1 = (A,B)

Create R2 = R – (B – A)

Note that: R1 ∩ R2 = A and A → AB (= R1), so this is lossless decomposition

Repeat for *R1, and R2*

Example 1

R = (A, B, C)

F = {A → B, B → C}

Candidate keys = {A}

BCNF = No. B → C violates.

B → C

R1 = (B, C)

F1 = {B → C}

Candidate keys = {B}

BCNF = true

R2 = (A, B)

F2 = {A → B}

Candidate keys = {A}

BCNF = true

Example 2-1

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

F = {A → B, BC → D}

Candidate keys = {ACE}

BCNF = Violated by {A → B, BC → D} etc…

A → B

From A → B and BC → D by pseudo-transitivity

R1 = (A, B)

F1 = {A → B}

Candidate keys = {A}

BCNF = true

R2 = (A, C, D, E)

F2 = {AC → D}

Candidate keys = {ACE}

BCNF = false (AC → D)

AC → D

R3 = (A, C, D)

F3 = {AC → D}

Candidate keys = {AC}

BCNF = true

R4 = (A, C, E)

F4 = {} [[only trivial]]

Candidate keys = {ACE}

BCNF = true

Dependency preservation ???

We can check:

A → B (R1), AC → D (R3),

but we lost BC → D

So this is not a dependency

-preserving decomposition

Example 2-2

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

F = {$A \rightarrow B$, $BC \rightarrow D$}

Candidate keys = {ACE}

BCNF = Violated by {$A \rightarrow B$, $BC \rightarrow D$} etc…

$BC \rightarrow D$

R1 = (B, C, D)

F1 = {$BC \rightarrow D$}

Candidate keys = {BC}

BCNF = true

R2 = (B, C, A, E)

F2 = {$A \rightarrow B$}

Candidate keys = {ACE}

BCNF = false ($A \rightarrow B$)

$A \rightarrow B$

R3 = (A, B)

F3 = {$A \rightarrow B$}

Candidate keys = {A}

BCNF = true

R4 = (A, C, E)

F4 = {} [[only trivial]]

Candidate keys = {ACE}

BCNF = true

Dependency preservation ???

We can check:

$BC \rightarrow D$ (R1), $A \rightarrow B$ (R3),

Dependency-preserving decomposition

BCNF may not preserve dependencies

- Not always possible to find a dependency-preserving decomposition that is in BCNF.
- NP-Hard to find one if it exists

BCNF and redundancy

MovieTitle	MovieYear	StarName	Address
Star wars	1977	Harrison Ford	Address 1, LA
Star wars	1977	Harrison Ford	Address 2, FL
Indiana Jones	198x	Harrison Ford	Address 1, LA
Indiana Jones	198x	Harrison Ford	Address 2, FL
Witness	19xx	Harrison Ford	Address 1, LA
Witness	19xx	Harrison Ford	Address 2, FL
…	…	…	…

Lot of redundancy

FDs ? No non-trivial FDs.

So the schema is trivially in BCNF

What went wrong ?

Multi-valued Dependencies

- The redundancy is because of *multi-valued dependencies*
- *Denoted:*

  $$starname \twoheadrightarrow address$$

- Should not happen if the schema is constructed from an E/R diagram

How to deal with these issues

- 3NF is an alternative to BCNF that does preserves dependencies
  - But may allow some FD redundancy
- 4NF removes redundancies due to multi-valued dependencies
- See book (not in assigned reading for more details)
  - 4NF is typically desired and achieved
    - A good E/R diagram won’t generate non-4NF relations at all
  - Choice between 3NF and BCNF is up to the designer

Comparing the normal forms

	3NF	BCNF	4NF
Eliminates redundancy because of FD's	Mostly	Yes	Yes
Eliminates redundancy because of MVD's	No	No	Yes
Preserves FDs	Yes.	Maybe	Maybe
Preserves MVDs	Maybe	Maybe	Maybe

4NF is typically desired and achieved.

A good E/R diagram won't generate non-4NF relations at all

Choice between 3NF and BCNF is up to the designer

Database design process

- Three ways to come up with a schema

1. Using E/R diagram
   - If good, then little normalization is needed
   - Tends to generate 4NF designs
2. A universal relation *R* that contains all attributes.
   - Called universal relation approach
   - Note that MVDs will be needed in this case
3. An *ad hoc* schema that is then normalized
   - MVDs may be needed in this case

Recap

- What about 1st and 2nd normal forms ?
- 1NF:
  - Essentially says that no set-valued attributes allowed
  - Formally, a domain is called *atomic* if the elements of the domain are considered indivisible
  - A schema is in 1NF if the domains of all attributes are atomic
  - We assumed 1NF throughout the discussion
    - Non 1NF is just not a good idea
- 2NF:
  - Mainly historic interest
  - See Exercise 7.15 in the book

Addendum

- Denormalization
  - After doing the normalization, we may have too many tables
  - We may *denormalize* for performance reasons
    - Too many tables → too many joins during queries
  - A better option is to use *views* instead
    - So if a specific set of tables is joined often, create a view on the join

- More advanced normal forms
  - project-join normal form (PJNF or 5NF)
  - domain-key normal form
  - Rarely used in practice