CMU SCS

Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

Lecture #17: Schema Refinement & Normalization - Normal Forms

(R&G, ch. 19)

CMU SCS

Overview - detailed

- DB design and normalization
  - pitfalls of bad design
  - decomposition
  - normal forms

CMU SCS

Goal

- Design 'good' tables
  - sub-goal#1: define what 'good' means
  - sub-goal#2: fix 'bad' tables
- in short: "we want tables where the attributes depend on the primary key, on the **whole** key, and **nothing but** the key"
- Let's see why, and how:

CMU SCS

Pitfalls

takes1 (<u>ssn, c-id</u>, grade, name, address)

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main

CMU SCS

reminders

Pitfalls

'Bad' - why? because: ssn->address, name

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
123	211	A	smith	Main

CMU SCS

reminders

Pitfalls?

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
123	211	A	smith	Main

- Redundancy
  - ??
  - ??

CMU SCS

reminders

Pitfalls

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
123	211	A	smith	Main

- Redundancy
  - space
  - (inconsistencies)
  - insertion/deletion anomalies:

CMU SCS

reminders

Pitfalls

- insertion anomaly:
  - "jones" registers, but takes no class - no place to store his address!

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
…	…	…	…	…
234	null	null	jones	Forbes

CMU SCS

reminders

Pitfalls

- deletion anomaly:
  - delete the last record of 'smith' (we lose his address!)

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
123	211	A	smith	Main

CMU SCS

Solution: decomposition

- split offending table in two (or more), eg.:

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
123	211	A	smith	Main

? ?

CMU SCS

Overview - detailed

- DB design and normalization
  - pitfalls of bad design
  - decomposition
    - lossless join decomp.
    - dependency preserving
  - normal forms

CMU SCS

Decompositions

There are 'bad' decompositions. Good ones are:

- lossless and **MUST HAVE**
- dependency preserving **Nice to have**

CMU SCS

Decompositions - lossy:

R1(ssn, grade, name, address) R2(c-id, grade)

Ssn	Grade	Name	Address
123	A	smith	Main
123	B	smith	Main
234	A	jones	Forbes

c-id	Grade
413	A
415	B
211	A

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
234	211	A	jones	Forbes

ssn->name, address

ssn, c-id -> grade

Faloutsos & Pavlo CMU SCS 15-415/615 13

CMU SCS

Decompositions - lossy:

can not recover original table with a join!

Ssn	Grade	Name	Address
123	A	smith	Main
123	B	smith	Main
234	A	jones	Forbes

c-id	Grade
413	A
415	B
211	A

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
234	211	A	jones	Forbes

ssn->name, address

ssn, c-id -> grade

Faloutsos & Pavlo CMU SCS 15-415/615 14

CMU SCS

Decompositions - overview

There are ‘bad’ decompositions. Good ones are:

- lossless and **MUST HAVE**
- dependency preserving **Nice to have**

Faloutsos & Pavlo CMU SCS 15-415/615 15

CMU SCS

Decompositions

example of non-dependency preserving

S#	address	status
123	London	E
125	Paris	E
234	Pitts.	A

S# -> address, status

address -> status

S#	address
123	London
125	Paris
234	Pitts.

S#	status
123	E
125	E
234	A

S# -> address **S# -> status**

Faloutsos & Pavlo CMU SCS 15-415/615 16

Decompositions

(drill: is it lossless?)

S#	address	status
123	London	E
125	Paris	E
234	Pitts.	A

S# -> address, status

address -> status

S#	address
123	London
125	Paris
234	Pitts.

S#	status
123	E
125	E
234	A

S# -> address **S# -> status**

Decompositions - overview

There are 'bad' decompositions. Good ones are:

- #1) lossless and **MUST HAVE**
- #2) dependency preserving **Nice to have**

How to automatically determine #1 and #2?

Decompositions - lossless

Definition:

consider schema R, with FD 'F'. R1, R2 is a lossless join decomposition of R if we **always** have: $r1 \bowtie r2 = r$

An easier criterion?

Decomposition - lossless

Theorem: lossless join decomposition if the joining attribute is a superkey in at least one of the new tables

Formally:

$R1 \cap R2 \rightarrow R1 \; or$

$R1 \cap R2 \rightarrow R2$

CMU SCS

Decomposition - lossless

example:

R1

Ssn	c-id	Grade
123	413	A
123	415	B
234	211	A

ssn, c-id -> grade

R2

Ssn	Name	Address
123	smith	Main
234	jones	Forbes

ssn->name, address

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
234	211	A	jones	Forbes

ssn->name, address

ssn, c-id -> grade

Faloutsos & Pavlo CMU SCS 15-415/615 21

CMU SCS

Overview - detailed

- DB design and normalization
  - pitfalls of bad design
  - decomposition
    - lossless join decomp.
    - **dependency preserving**
  - normal forms

Faloutsos & Pavlo CMU SCS 15-415/615 22

CMU SCS

Decomposition - depend. pres.

informally: we don't want the original FDs to span two tables - counter-example:

S#	address	status
123	London	E
125	Paris	E
234	Pitts.	A

S# -> address, status

address -> status

S#	address
123	London
125	Paris
234	Pitts.

S#	status
123	E
125	E
234	A

S# -> address **S# -> status**

Faloutsos & Pavlo CMU SCS 15-415/615 23

CMU SCS

Decomposition - depend. pres.

informally: we don't want the original FDs to span two tables - counter-example:

S#	address	status
123	London	E
125	Paris	E
234	Pitts.	A

S# -> address, status

address -> status

S#	address
123	London
125	Paris
234	Pitts.

S#	status
123	E
125	E
234	A

S# -> address **S# -> status**

(Q: Why is it an issue?)

Faloutsos & Pavlo CMU SCS 15-415/615 24

CMU SCS

Decomposition - depend. pres.

informally: we don’t want the original FDs to span two tables - counter-example:

S#	address	status
123	London	E
125	Paris	E
234	Pitts.	A

S#	address
123	London
125	Paris
234	Pitts.
999	**Pitts.**

S#	status
123	E
125	E
234	A
999	**E**

S# -> address, status

address -> status

S# -> address **S# -> status**

(Q: Why is it an issue?)

(A: insert [999, Pitts., E])

Faloutsos & Pavlo CMU SCS 15-415/615 25

CMU SCS

Decomposition - depend. pres.

informally: we don’t want the original FDs to span two tables - counter-example:

S#	address	status
123	London	E
125	Paris	E
234	Pitts.	A

S#	address
123	London
125	Paris
234	Pitts.

S#	status
123	E
125	E
234	A

S# -> address, status

address -> status

S# -> address **S# -> status**

Faloutsos & Pavlo CMU SCS 15-415/615 26

CMU SCS

of the **COVER**

Decomposition - depend. pres.

informally: we don’t want the original FDs to span two tables - counter-example:

S#	address	status
123	London	E
125	Paris	E
234	Pitts.	A

S#	address
123	London
125	Paris
234	Pitts.

S#	status
123	E
125	E
234	A

S# -> address, status

address -> status

S# -> address **S# -> status**

Faloutsos & Pavlo CMU SCS 15-415/615 27

CMU SCS

Decomposition - depend. pres.

- A subtle point
- To avoid it, use the ‘canonical cover’ of the FDs

Faloutsos & Pavlo CMU SCS 15-415/615 28

CMU SCS

Decomposition - depend. pres.

dependency preserving decomposition:

S#	address	status
123	London	E
125	Paris	E
234	Pitts.	A

S# -> address, status

address -> status

S#	address
123	London
125	Paris
234	Pitts.

address	status
London	E
Paris	E
Pitts.	A

S# -> address **address -> status**

(but: S#->status ?)

CMU SCS

Decomposition - depend. pres.

informally: we don't want the original FDs to span two tables.

More specifically: … the FDs of the **canonical cover**.

CMU SCS

Decomposition - depend. pres.

Q: why is dependency preservation good?

S#	address
123	London
125	Paris
234	Pitts.

S#	status
123	E
125	E
234	A

S# -> address

S# -> status

(address->status: 'lost')

S#	address
123	London
125	Paris
234	Pitts.

address	status
London	E
Paris	E
Pitts.	A

S# -> address **address -> status**

CMU SCS

Decomposition - depend. pres.

A1: insert [999, Pitts., E] -> REJECT

S#	address
123	London
125	Paris
234	Pitts.
999	**Pitts.**

S#	status
123	E
125	E
234	A
999	**E**

S# -> address

S# -> status

(address->status: 'lost')

S#	address
123	London
125	Paris
234	Pitts.
999	**Pitts.**

address	status
London	E
Paris	E
Pitts.	A
Pitts.	**E**

✖

S# -> address **address -> status**

CMU SCS

Decomposition - depend. pres.

A2: eg., record that 'Philly' has status 'A'

S#	address
123	London
125	Paris
234	Pitts.
null	**Philly**

S#	status
123	E
125	E
234	A
null	**A**

S# -> address

S# -> status

(address->status: 'lost')

S#	address
123	London
125	Paris
234	Pitts.

address	status
London	E
Paris	E
Pitts.	A
Philly	**A**

✔

S# -> address **address -> status**

Faloutsos & Pavlo CMU SCS 15-415/615 33

CMU SCS

Decomposition - conclusions

- decompositions should always be lossless
  - joining attribute -> superkey **MUST HAVE**
- whenever possible, we want them to be dependency preserving (occasionally, impossible - see 'STJ' example later…)

Nice to have

Faloutsos & Pavlo CMU SCS 15-415/615 34

CMU SCS

Overview - detailed

- DB design and normalization
  - pitfalls of bad design
  - decomposition (-> how to fix the problem)
  - ➡ **normal forms** (-> how to detect the problem)
    - BCNF,
    - 3NF
    - (1NF, 2NF)

Faloutsos & Pavlo CMU SCS 15-415/615 35

CMU SCS

Normal forms - BCNF

We saw how to fix 'bad' schemas -
but what is a 'good' schema?

Answer: 'good', if it obeys a 'normal form',
ie., a set of rules.

Typically: Boyce-Codd Normal form

Faloutsos & Pavlo CMU SCS 15-415/615 36

CMU SCS

Normal forms - BCNF

Defn.: Rel. *R* is in BCNF wrt *F*, if

- informally: everything depends on the full key, and nothing but the key

Or

- semi-formally: *every determinant (of the cover) is a candidate key*

Faloutsos & Pavlo CMU SCS 15-415/615 37

CMU SCS

Normal forms - BCNF

Example and counter-example:

Ssn	Name	Address
123	smith	Main
999	smith	Shady
234	jones	Forbes

ssn->name, address

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
234	211	A	jones	Forbes

ssn->name, address

ssn, c-id -> grade

Faloutsos & Pavlo CMU SCS 15-415/615 38

CMU SCS

Normal forms - BCNF

Formally: for every FD *a->b* in F

- *a->b* is trivial (*a* superset of *b*) or
- *a* is a superkey

Faloutsos & Pavlo CMU SCS 15-415/615 39

CMU SCS

Normal forms

Drill: Check formal dfn:

- *a->b* trivial, or
- *a* is superkey

Example and counter-example:

Ssn	Name	Address
123	smith	Main
999	smith	Shady
234	jones	Forbes

ssn->name, address

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
234	211	A	jones	Forbes

ssn->name, address

ssn, c-id -> grade

Faloutsos & Pavlo CMU SCS 15-415/615 40

CMU SCS

Normal forms

Drill: Check formal dfn:
- *a->b* trivial, or
- *a* is superkey

Example and counter-example:

Ssn	Name	Address
123	smith	Main
999	smith	Shady
234	jones	Forbes

ssn->name, address

ssn,name -> address

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
234	211	A	jones	Forbes

ssn->name, address

ssn, c-id -> grade

ssn, name -> address

ssn, c-id, name -> grade

CMU SCS

Normal forms - BCNF

Theorem: given a schema R and a set of FD 'F', we can **always** decompose it to schemas R1, … Rn, so that
- R1, … Rn are in **BCNF** and
- the decompositions are **lossless**.

(but, some decomp. might lose dependencies)

CMU SCS

Normal forms - BCNF

How? algorithm in book: for a relation R

- for every FD X->A that violates BCNF, decompose to tables (X,A) and (R-A)
- repeat recursively

eg. TAKES1(ssn, c-id, grade, name, address)

ssn -> name, address

ssn, c-id -> grade

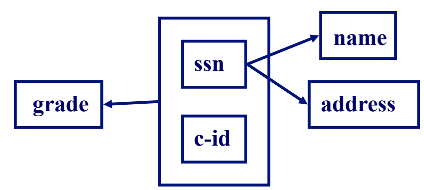
CMU SCS

not in book

Normal forms - BCNF

eg. TAKES1(ssn, c-id, grade, name, address)

ssn -> name, address ssn, c-id -> grade



CMU SCS

Normal forms - BCNF

Ssn	c-id	Grade
123	413	A
123	415	B
234	211	A

ssn, c-id -> grade

Ssn	Name	Address
123	smith	Main
123	smith	Main
234	jones	Forbes

ssn->name, address

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
234	211	A	jones	Forbes

ssn->name, address

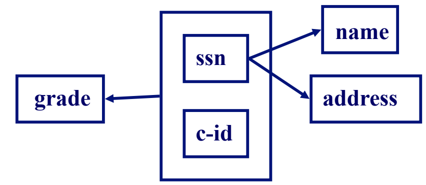
ssn, c-id -> grade

CMU SCS

not in book

Normal forms - BCNF

pictorially: we want a 'star' shape



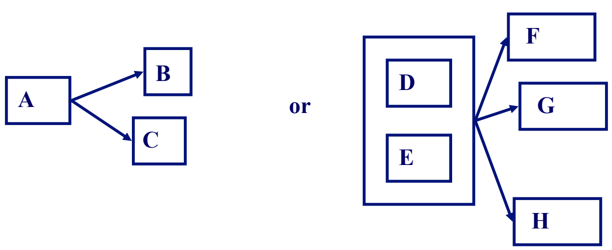
:not in BCNF

CMU SCS

not in book

Normal forms - BCNF

pictorially: we want a 'star' shape



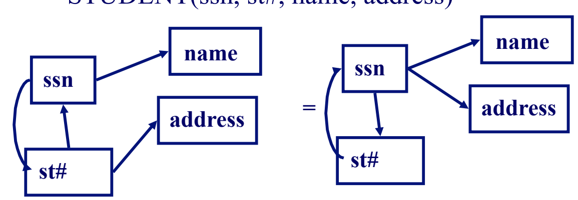
CMU SCS

not in book

Normal forms - BCNF

or a star-like: (eg., 2 cand. keys):

STUDENT(ssn, st#, name, address)

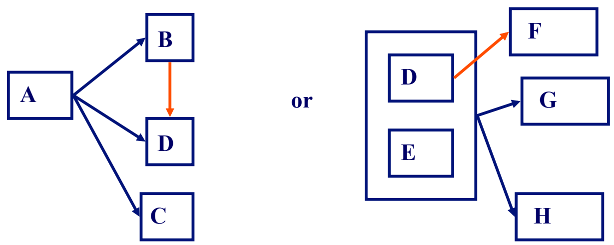


CMU SCS

not in book

Normal forms - BCNF

but **not**:



Faloutsos & Pavlo CMU SCS 15-415/615 49

CMU SCS

Overview - detailed

- DB design and normalization
  - pitfalls of bad design
  - decomposition (-> how to fix the problem)
  - **normal forms** (-> how to detect the problem)
    - BCNF,
    - 3NF
    - (1NF, 2NF)

Faloutsos & Pavlo CMU SCS 15-415/615 50

CMU SCS

Reminder

Normal forms - BCNF

Theorem: given a schema R and a set of FD 'F', we can **always** decompose it to schemas R1, … Rn, so that
- R1, … Rn are in **BCNF** and
- the decompositions are **lossless**.

(but, some decomp. might lose dependencies)

Faloutsos & Pavlo CMU SCS 15-415/615 51

CMU SCS

Reminder

Normal forms - BCNF

Theorem: given a schema R and a set of FD 'F', we can **always** decompose it to schemas R1, … Rn, so that
- R1, … Rn are in **BCNF** and
- the decompositions are **lossless**.

(but, some decomp. might lose dependencies)

How is this possible?

Faloutsos & Pavlo CMU SCS 15-415/615 52

Subtle answer

In some rare cases, like the

(Student, Teacher, subJect)

setting:

Normal forms - 3NF

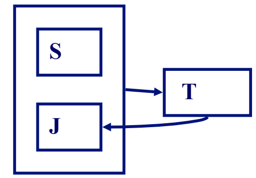
consider the 'classic' case:

STJ(Student, Teacher, subJect)

T-> J

S,J -> T

is it BCNF?

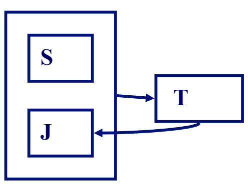


Normal forms - 3NF

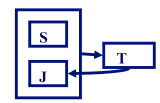
STJ(Student, Teacher, subJect)

T-> J S,J -> T

How to decompose it to BCNF?



Normal forms - 3NF



STJ(Student, Teacher, subJect)

T-> J S,J -> T

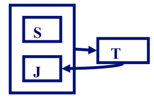
1) R1(T,J) R2(S,J)

(BCNF? - lossless? - dep. pres.?)

2) R1(T,J) R2(S,T)

(BCNF? - lossless? - dep. pres.?)

Normal forms - 3NF



STJ(Student, Teacher, subJect)

T-> J S,J -> T

1) R1(T,J) R2(S,J)

(BCNF? **Y+Y** - lossless? **N** - dep. pres.? **N**)

2) R1(T,J) R2(S,T)

(BCNF? **Y+Y** - lossless? **Y** - dep. pres.? **N**)

Normal forms - 3NF

STJ(Student, Teacher, subJect)

T-> J S,J -> T

in this case: impossible to have both

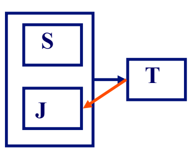
- BCNF **and**
- dependency preservation

Welcome 3NF!

Normal forms - 3NF

STJ(Student, Teacher, subJect)

T-> J S,J -> T

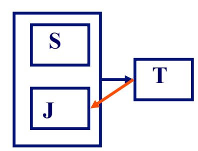


informally, 3NF 'forgives' the red arrow in the canonical cover

Normal forms - 3NF

STJ(Student, Teacher, subJect)

T-> J S,J -> T



Formally, a rel. R with FDs 'F' is in 3NF if: for every *a->b* in F:

- it is trivial or
- *a* is a superkey or
- *b*: part of a candidate key

Normal forms - 3NF

how to bring a schema to 3NF?

two algo's in book: First one:

- start from ER diagram and turn to tables
- then we have a set of tables R1, ... Rn which are in 3NF
- for each FD (X->A) in the cover that is not preserved, create a table (X,A)

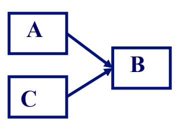
Normal forms - 3NF

how to bring a schema to 3NF?

two algo's in book: Second one ('synthesis')

- take all attributes of R
- for each FD (X->A) in the cover, add a table (X,A)
- if not lossless, add a table with appropriate key

Normal forms - 3NF



Example:

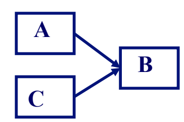
R: ABC

F: A->B, C->B

Q1: what is the cover? What is the cand. key?

Q2: what is the decomposition to 3NF?

Normal forms - 3NF



Example:

R: ABC

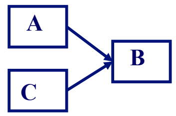
F: A->B, C->B

Q1: what is the cover? What is the cand. key?

A1: 'F' is the cover; 'AC' is the cand. key

Q2: what is the decomposition to 3NF?

Normal forms - 3NF



Example:

R: ABC

F: A->B, C->B

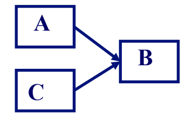
Q1: what is the cover? What is the cand. key?

A1: 'F' is the cover; 'AC' is the cand. key

Q2: what is the decomposition to 3NF?

A2: R1(A,B), R2(C,B), ... [is it lossless??]

Normal forms - 3NF



Example:

R: ABC

F: A->B, C->B

Q1: what is the cover? What is the cand. key?

A1: 'F' is the cover; 'AC' is the cand. key

Q2: what is the decomposition to 3NF?

A2: R1(A,B), R2(C,B), R3(A,C)

Normal forms - 3NF vs BCNF

- If 'R' is in BCNF, it is always in 3NF (but not the reverse)
- In practice, aim for
  - BCNF; lossless join; and dep. preservation
- if impossible, we accept
  - 3NF; but insist on lossless join and dep. preservation

Normal forms - more details

- why '3'NF? what is 2NF? 1NF?
- 1NF: attributes are atomic (ie., no set-valued attr., a.k.a. 'repeating groups')

Ssn	Name	Dependents
123	Smith	Peter Mary John
234	Jones	Ann Michael

not 1NF

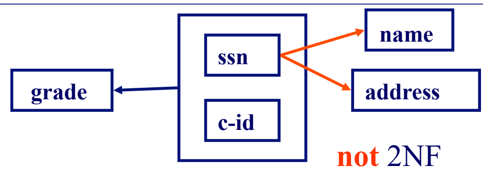
CMU SCS

Normal forms - more details

2NF: 1NF and non-key attr. fully depend on the key

counter-example: TAKES1(ssn, c-id, grade, name, address)

ssn -> name, address   ssn, c-id -> grade



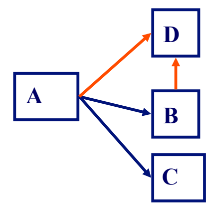
not 2NF

Faloutsos & Pavlo   CMU SCS 15-415/615   69

CMU SCS

Normal forms - more details

- 3NF: 2NF and no transitive dependencies
- counter-example:



in 2NF, but **not** in 3NF

Faloutsos & Pavlo   CMU SCS 15-415/615   70

CMU SCS

Normal forms - more details

- 4NF, multivalued dependencies etc: IGNORE
- in practice, E-R diagrams usually lead to tables in BCNF

Faloutsos & Pavlo   CMU SCS 15-415/615   71

CMU SCS

Overview - conclusions

DB design and normalization

- pitfalls of bad design
- decompositions (lossless, dep. preserving)
- normal forms (BCNF or 3NF)

"everything should depend on the key, the **whole** key, and **nothing but** the key"

Faloutsos & Pavlo   CMU SCS 15-415/615   72