Carnegie Mellon Univ.
Dept. of Computer Science
15-415 - Database Applications

Schema Refinement & Normalization - Functional Dependencies

(R&G, ch. 19)

Functional dependencies

- motivation: ‘good’ tables

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

‘good’ or ‘bad’?

Functional dependencies

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

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

Functional dependencies

'Bad' – Q: why?

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

Functional Dependencies

- A: Redundancy
  - space
  - inconsistencies
  - insertion/deletion anomalies (later…)
- Q: What caused the problem?

Functional dependencies

- A: ‘name’ depends on the ‘ssn’
- define ‘depends’

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

Overview

- Functional dependencies
  - why
  - definition
  - Armstrong's "axioms"
  - closure and cover

Functional dependencies

Definition: $a \rightarrow b$

'a' functionally determines 'b'

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

Functional dependencies

Informally: ‘if you know ‘a’, there is only one ‘b’ to match’

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

Functional dependencies

formally:

$$X \rightarrow Y \qquad \Rightarrow \quad (t1[x] = t2[x] \Rightarrow t1[y] = t2[y])$$

if two tuples agree on the 'X' attribute,
the *must* agree on the 'Y' attribute, too
(eg., if ssn is the same, so should address)

Functional dependencies

- ‘X’, ‘Y’ can be **sets** of attributes
- Q: other examples??

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

Functional dependencies

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

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

Overview

- Functional dependencies
  - why
  - definition
  - Armstrong's "axioms"
  - closure and cover

Functional dependencies

Closure of a set of FD: all implied FDs - eg.:

ssn -> name, address

ssn, c-id -> grade

imply

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

ssn, c-id -> ssn

FDs - Armstrong's axioms

Closure of a set of FD: all implied FDs - eg.:

ssn -> name, address

ssn, c-id -> grade

how to find all the implied ones, systematically?

FDs - Armstrong's axioms

"Armstrong's axioms" guarantee soundness and completeness:

- Reflexivity: $Y \subseteq X \Rightarrow X \rightarrow Y$

  eg., ssn, name -> ssn

- Augmentation $X \rightarrow Y \Rightarrow XW \rightarrow YW$

  eg., ssn->name then ssn,grade-> name,grade

FDs - Armstrong's axioms

- Transitivity

$$\left.\begin{array}{l} X \rightarrow Y \\ Y \rightarrow Z \end{array}\right\} \Rightarrow X \rightarrow Z$$

ssn -> address

address -> county-tax-rate

THEN:

ssn -> county-tax-rate

FDs - Armstrong's axioms

Reflexivity: $Y \subseteq X \Rightarrow X \rightarrow Y$

Augmentation: $X \rightarrow Y \Rightarrow XW \rightarrow YW$

Transitivity:

$$\left.\begin{array}{l} X \rightarrow Y \\ Y \rightarrow Z \end{array}\right\} \Rightarrow X \rightarrow Z$$

'sound' and 'complete'

FDs - Armstrong's axioms

Additional rules:

- Union
$$\left.\begin{matrix} X \rightarrow Y \\ X \rightarrow Z \end{matrix}\right\} \Rightarrow X \rightarrow YZ$$
- Decomposition
$$X \rightarrow YZ \Rightarrow \left.\begin{matrix} X \rightarrow Y \\ X \rightarrow Z \end{matrix}\right\}$$
- Pseudo-transitivity
$$\left.\begin{matrix} X \rightarrow Y \\ YW \rightarrow Z \end{matrix}\right\} \Rightarrow XW \rightarrow Z$$

FDs - Armstrong's axioms

Prove 'Union' from three axioms:

$$\left.\begin{array}{l} X \rightarrow Y \\ X \rightarrow Z \end{array}\right\} \overset{?}{\Rightarrow} X \rightarrow YZ$$

FDs - Armstrong's axioms

Prove 'Union' from three axioms:

$$\left.\begin{array}{ll} X \rightarrow Y & (1) \\ X \rightarrow Z & (2) \end{array}\right\}$$

$$(1) + augm.\, w/\, Z \Rightarrow XZ \rightarrow YZ \; (3)$$

$$(2) + augm.\, w/\, X \Rightarrow XX \rightarrow XZ \; (4)$$

$$but \;\; XX \;\; is \;\; X;\, thus$$

$$(3) + (4) \; and \; transitivity \Rightarrow X \rightarrow YZ$$

FDs - Armstrong's axioms

Prove Pseudo-transitivity:

$$Y \subseteq X \Rightarrow X \rightarrow Y$$

$$X \rightarrow Y \Rightarrow XW \rightarrow YW$$

$$\left.\begin{matrix} X \rightarrow Y \\ Y \rightarrow Z \end{matrix}\right\} \Rightarrow X \rightarrow Z$$

$$\left.\begin{matrix} X \rightarrow Y \\ YW \rightarrow Z \end{matrix}\right\} \overset{?}{\Rightarrow} XW \rightarrow Z$$

FDs - Armstrong's axioms

Prove Decomposition

$$Y \subseteq X \Rightarrow X \rightarrow Y$$

$$X \rightarrow Y \Rightarrow XW \rightarrow YW$$

$$\left.\begin{array}{l} X \rightarrow Y \\ Y \rightarrow Z \end{array}\right\} \Rightarrow X \rightarrow Z$$

$$X \rightarrow YZ \stackrel{?}{\Rightarrow} \left.\begin{array}{l} X \rightarrow Y \\ X \rightarrow Z \end{array}\right\}$$

Overview

- Functional dependencies
  - why
  - definition
  - Armstrong's "axioms"
  - closure and cover

FDs - Closure F+

Given a set F of FD (on a schema)

F+ is the set of all implied FD. Eg.,

takes(ssn, c-id, grade, name, address)

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

} F

FDs - Closure F+

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

F+

FDs - Closure A+

Given a set F of FD (on a schema)

A+ is the set of all attributes determined by A:

takes(ssn, c-id, grade, name, address)

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

{ssn}+ =??

FDs - Closure A+

takes(ssn, c-id, grade, name, address)

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

{ssn}+ ={ssn,
name, address }

FDs - Closure A+

takes(ssn, c-id, grade, name, address)

ssn, c-id -> grade

ssn-> name, address

} F

{c-id}+ = ??

FDs - Closure A+

takes(ssn, c-id, grade, name, address)

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

} F

{c-id, ssn}+ = ??

FDs - Closure A+

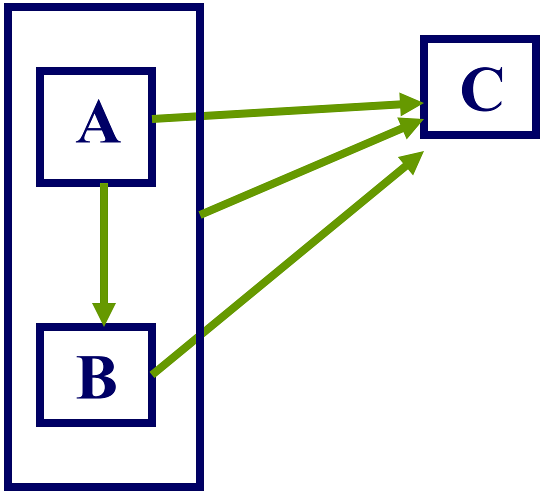
if A+ = {all attributes of table}

then ‘A’ is a **superkey**

FDs - A+ closure - not in book

Diagrams

AB->C (1)
A->BC (2)
B->C (3)
A->B (4)



FDs - ‘canonical cover’ Fc

Given a set F of FD (on a schema)

Fc is a minimal set of equivalent FD. Eg.,

takes(ssn, c-id, grade, name, address)

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

F

FDs - ‘canonical cover’ Fc

Fc

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

F

FDs - ‘canonical cover’ Fc

- why do we need it?
- define it properly
- compute it efficiently

FDs - ‘canonical cover’ Fc

- why do we need it?
  - easier to compute candidate keys
- define it properly
- compute it efficiently

FDs - ‘canonical cover’ Fc

- define it properly - three properties
  - 1) the RHS of every FD is a single attribute
  - 2) the closure of Fc is identical to the closure of F (ie., Fc and F are equivalent)
  - 3) Fc is minimal (ie., if we eliminate any attribute from the LHS or RHS of a FD, property #2 is violated

FDs - ‘canonical cover’ Fc

#3: we need to eliminate ‘extraneous’ attributes. An attribute is ‘extraneous if

- the closure is the same, before and after its elimination
- or if F-before implies F-after and vice-versa

FDs - ‘canonical cover’ Fc

ssn, c-id -> grade

ssn-> name, address

ssn,~~name~~-> name, address

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

F

FDs - ‘canonical cover’ Fc

Algorithm:

- examine each FD; drop extraneous LHS or RHS attributes; or redundant FDs
- make sure that FDs have a single attribute in their RHS
- repeat until no change

FDs - ‘canonical cover’ Fc

Trace algo for

AB->C (1)

A->BC (2)

B->C (3)

A->B (4)

FDs - ‘canonical cover’ Fc

Trace algo for

AB->C (1)

A->BC (2)

B->C (3)

A->B (4)

split (2):

AB->C (1)

A->B (2’)

A->C (2”)

B->C (3)

A->B (4)

FDs - ‘canonical cover’ Fc

AB->C (1)
~~A->B (2’)~~
A->C (2”)
B->C (3)
A->B (4)

AB->C (1)

A->C (2”)
B->C (3)
A->B (4)

FDs - ‘canonical cover’ Fc

AB->C (1)

A->C (2”)
B->C (3)
A->B (4)

(2”): redundant (implied by (4), (3) and transitivity

AB->C (1)

B->C (3)
A->B (4)

FDs - ‘canonical cover’ Fc

AB->C (1)	B->C (1’)
B->C (3)	B->C (3)
A->B (4)	A->B (4)

in (1), ‘A’ is extraneous:
(1),(3),(4) imply
(1’),(3),(4), and vice versa

FDs - 'canonical cover' Fc

~~B->C (1')~~

B->C (3)
A->B (4)

B->C (3)
A->B (4)

- **nothing is extraneous**
- **all RHS are single attributes**
- **final and original set of FDs are equivalent (same closure)**

FDs - ‘canonical cover’ Fc

BEFORE

AB->C (1)
A->BC (2)
B->C (3)
A->B (4)

AFTER

B->C (3)
A->B (4)

Overview - conclusions

- Functional dependencies
  - why
  - definition
  - Armstrong's "axioms"
  - closure and cover

Overview - detailed

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

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:

Pitfalls

takes1 (ssn, c-id, grade, name, address)

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

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

Pitfalls

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

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

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

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

? ?

Overview - detailed

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

Decompositions

There are ‘bad’ decompositions. Good ones are:

- lossless and
- dependency preserving

Decompositions - lossy:

R1(ssn, grade, name, address)

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

R2(c-id, grade)

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

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

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

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 - 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:

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

$$R1 \cap R2 \rightarrow R2$$

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

Overview - detailed

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

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

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 ?)

Decomposition - depend. pres.

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

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

Decomposition - depend. pres.

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**

Decomposition - depend. pres.

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

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**

Decomposition - conclusions

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

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)

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

Normal forms - BCNF

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

- informally: everything depends on the full key, and nothing but the key
- semi-formally: every determinant (of the cover) is a candidate key

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

Normal forms - BCNF

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

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

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)

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)

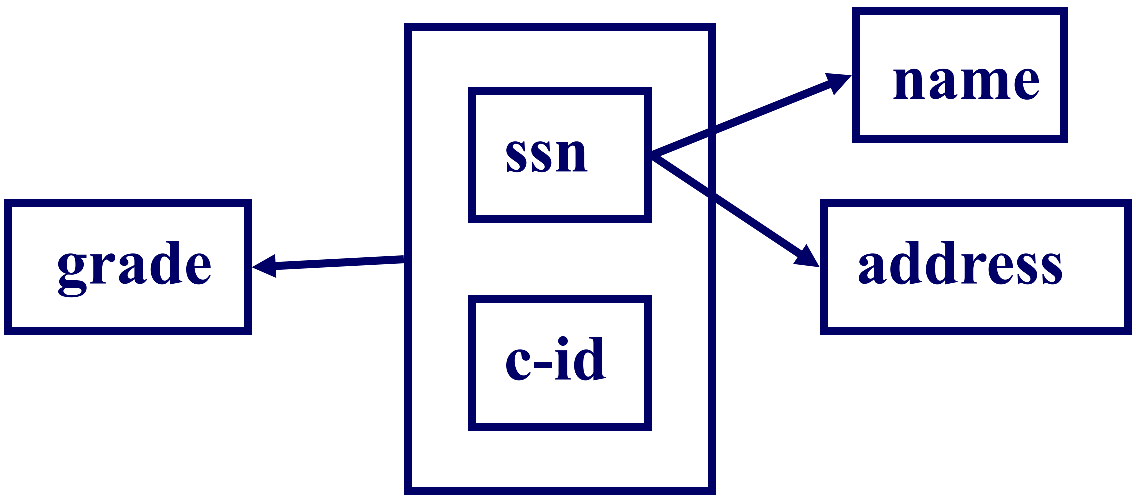
ssn -> name, address

ssn, c-id -> grade

Normal forms - BCNF

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

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



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

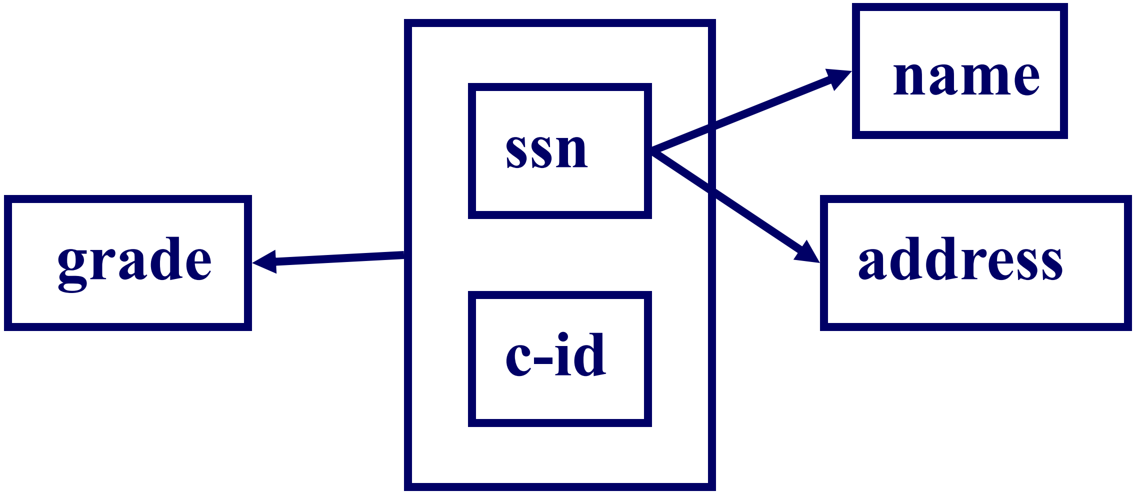
Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
234	211	A	jones	Forbes

ssn->name, address

ssn, c-id -> grade

Normal forms - BCNF

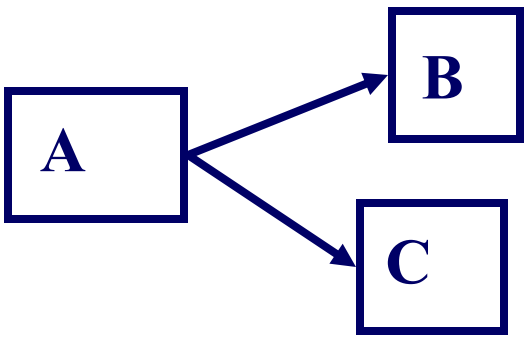
pictorially: we want a 'star' shape



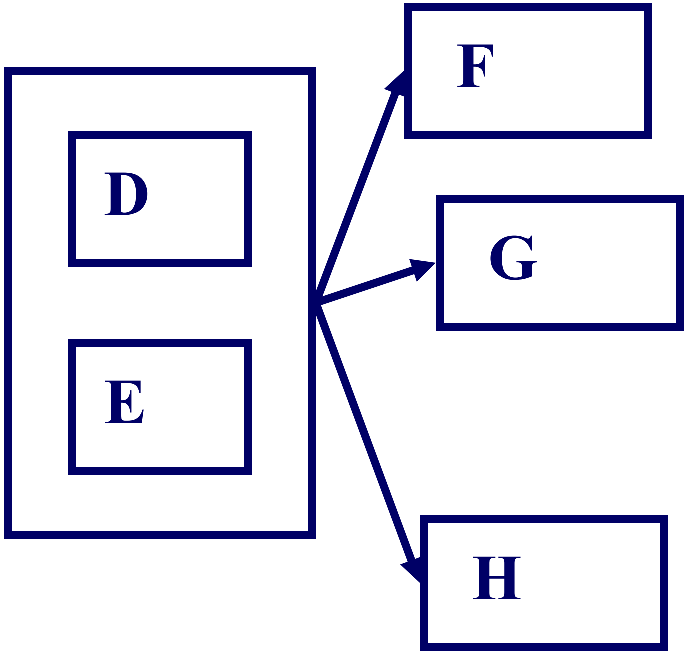
:not in BCNF

Normal forms - BCNF

pictorially: we want a 'star' shape



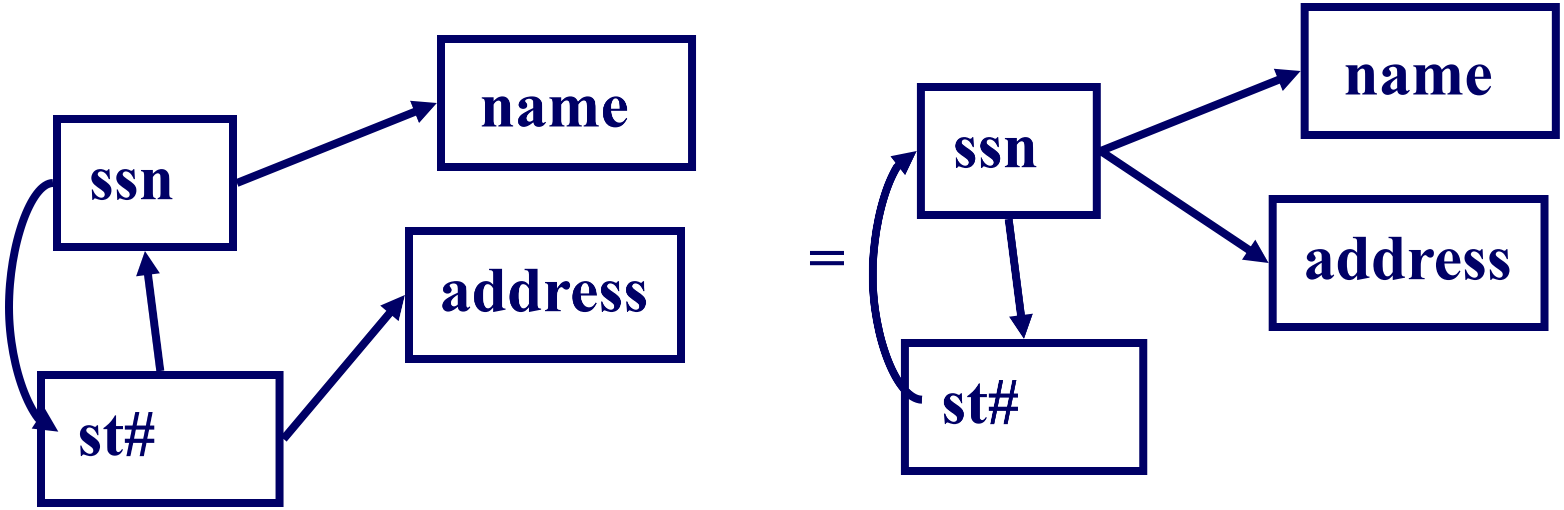
or



Normal forms - BCNF

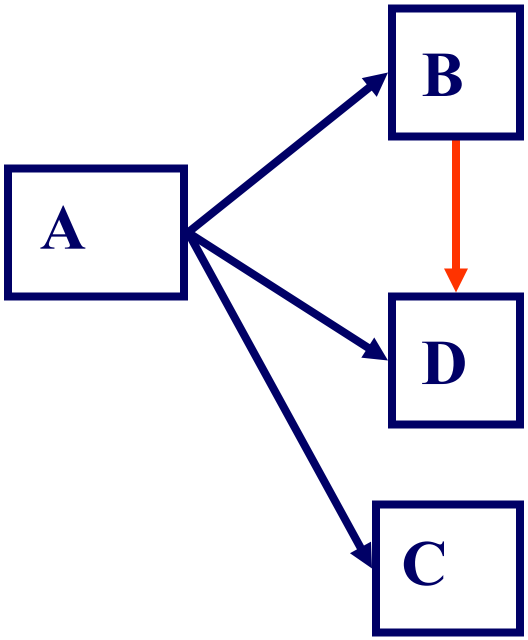
or a star-like: (eg., 2 cand. keys):

STUDENT(ssn, st#, name, address)

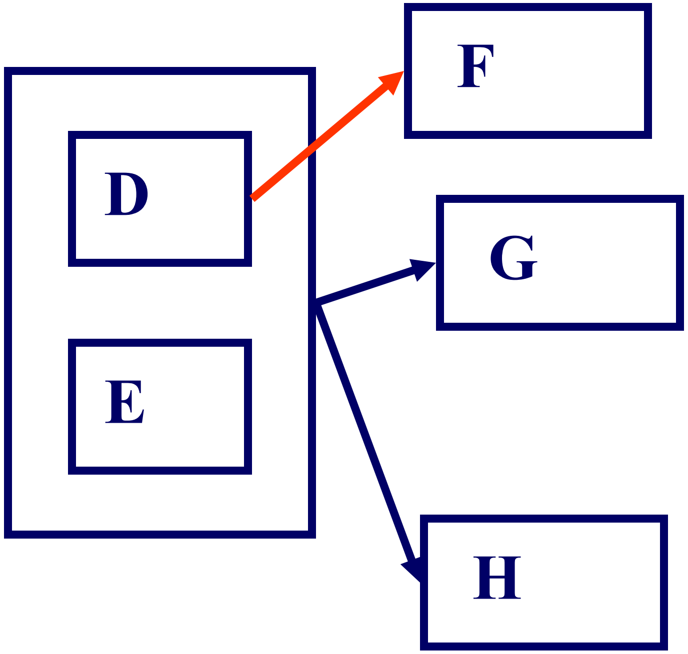


Normal forms - BCNF

but **not**:



or



Normal forms - 3NF

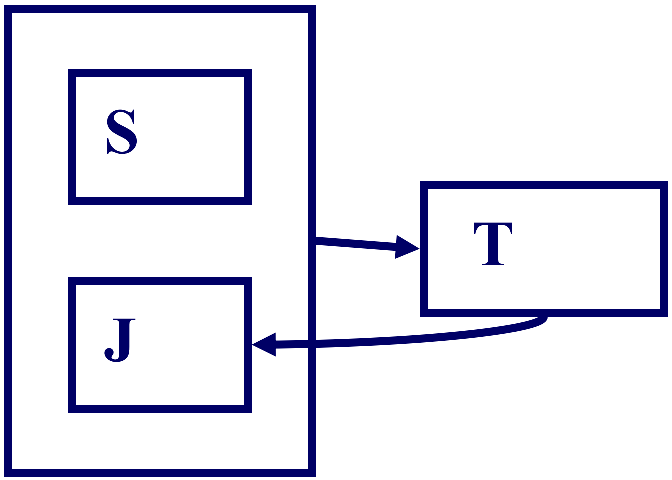
consider the 'classic' case:

STJ(Student, Teacher, subJect)

T-> J

S,J -> T

is it BCNF?

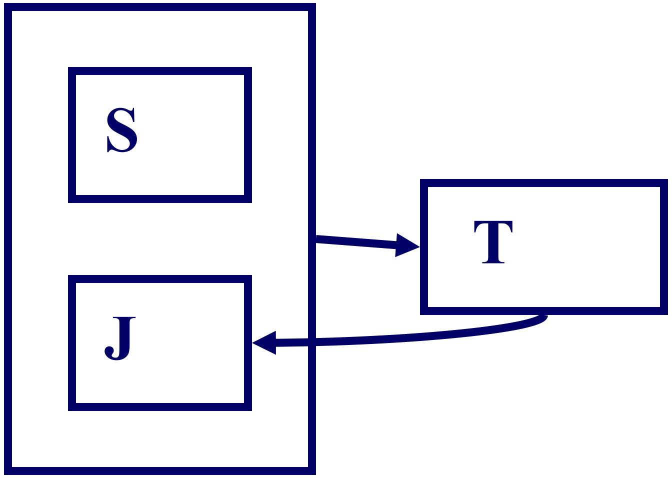


Normal forms - 3NF

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

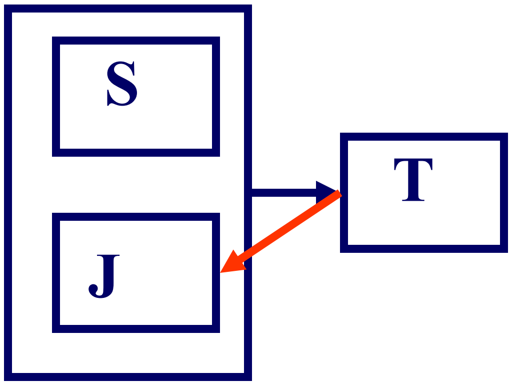
- BCNF **and**
- dependency preservation

Welcome 3NF!

Normal forms - 3NF

STJ(Student, Teacher, subJect)

T-> J S,J -> T

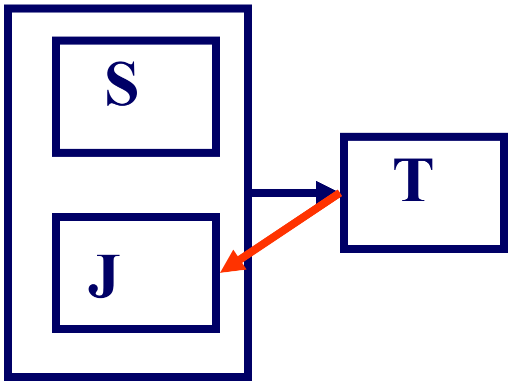


informally, 3NF 'forgives' the red arrow in the can. cover

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?

Q2: what is the decomposition to 3NF?

Normal forms - 3NF

Example:

R: ABC

F: A->B, C->B

Q1: what is the cover?

A1: ‘F’ is the cover

Q2: what is the decomposition to 3NF?

Normal forms - 3NF

Example:

R: ABC

F: A->B, C->B

Q1: what is the cover?

A1: ‘F’ is the cover

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?

A1: ‘F’ is the cover

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

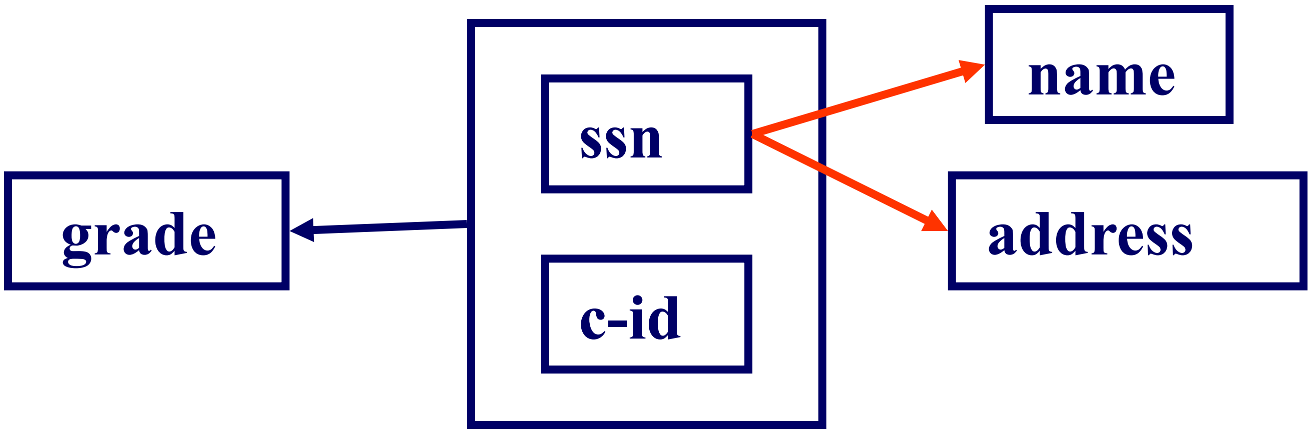
not 1NF

Normal forms - more details

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

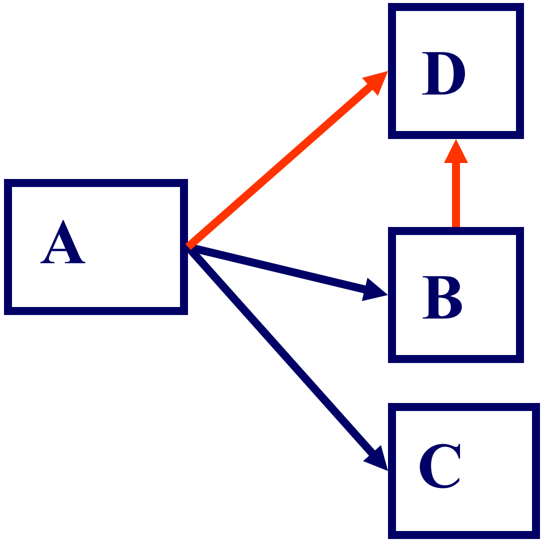
counter-example: TAKES1(ssn, c-id, grade, name, address)

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



Normal forms - more details

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



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

Normal forms - more details

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

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"