VirginiaTech

CS 4604: Introduction to Database Management Systems

B. Aditya Prakash

Lecture #14: BCNF, 3NF and Normalization

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)   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

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	Blacks.	A

S# -> address, status
address -> status

S#	address
123	London
125	Paris
234	Blacks.

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	Blacks.	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: if you are decomposing R into R1 and R2 then (so R = R1 U R2)

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

$$R1 \cap R2 \rightarrow R2$$

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	Blacks.	A

S# -> address, status
address -> status

S#	address
123	London
125	Paris
234	Blacks.

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	Blacks.	A

S# -> address, status
address -> status

S#	address
123	London
125	Paris
234	Blacks.

address	status
London	E
Paris	E
Blacks.	A

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

(but: S#->status ?)

Decomposition - depend. pres.

- informally: we don't want the original FDs to span two tables.
- So more specifically: … the FDs of the canonical cover.

Decomposition - depend. pres.

- why is dependency preservation good?

S#	address
123	London
125	Paris
234	Blacks.

S#	status
123	E
125	E
234	A

S# -> address

S# -> status

(address->status: 'lost')

S#	address
123	London
125	Paris
234	Blacks.

address	status
London	E
Paris	E
Blacks.	A

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

Decomposition - depend. pres.

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

S#	address
123	London
125	Paris
234	Blacks.

S#	status
123	E
125	E
234	A

S# -> address

S# -> status

(address->status: 'lost')

S#	address
123	London
125	Paris
234	Blacks.

address	status
London	E
Paris	E
Blacks.	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

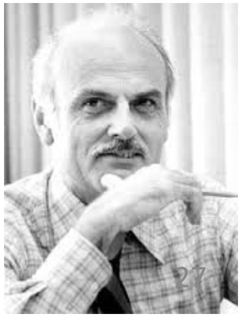
- DB design and normalization
  - pitfalls of bad design
  - decomposition (-> how to fix the problem)
  - **normal forms (-> how to detect the problem)**
    - **BCNF,**
    - **3NF**
    - **(1NF, 2NF)**

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 i.e the left-side (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 in S that violates BCNF, decompose to tables (X,A) and (R-A)

- repeat recursively

Q: how to get the FDs for the new relations (X, A) and (R-A)?

Ans: just project the FDs into them i.e. which FDs are in S and involve only attrs. of (X-A) (similarly for R-A)

Normal forms - BCNF

- How? algorithm in book: for a relation R

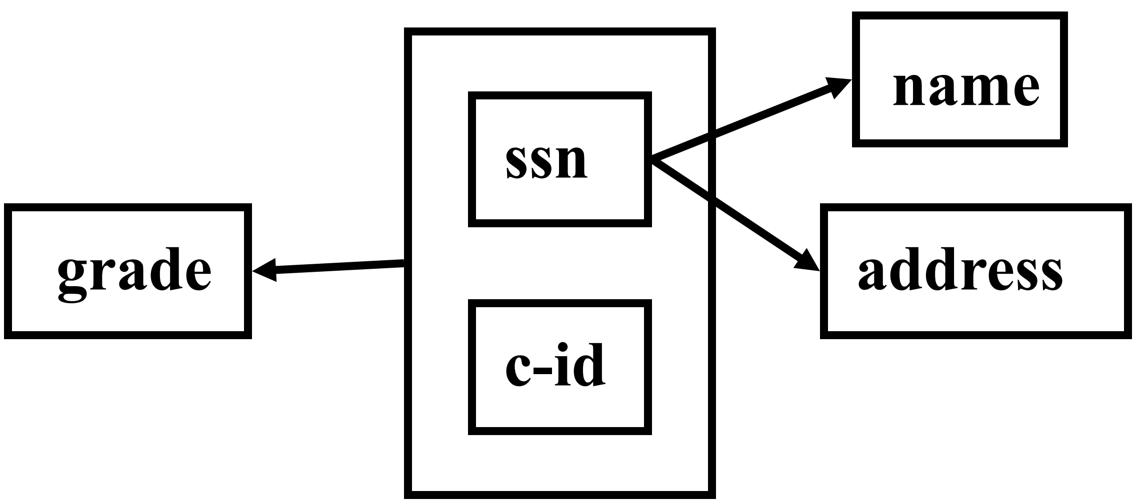
- for every FD X->A that violates BCNF, decompose to tables (X,A) and (R-A)

- 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

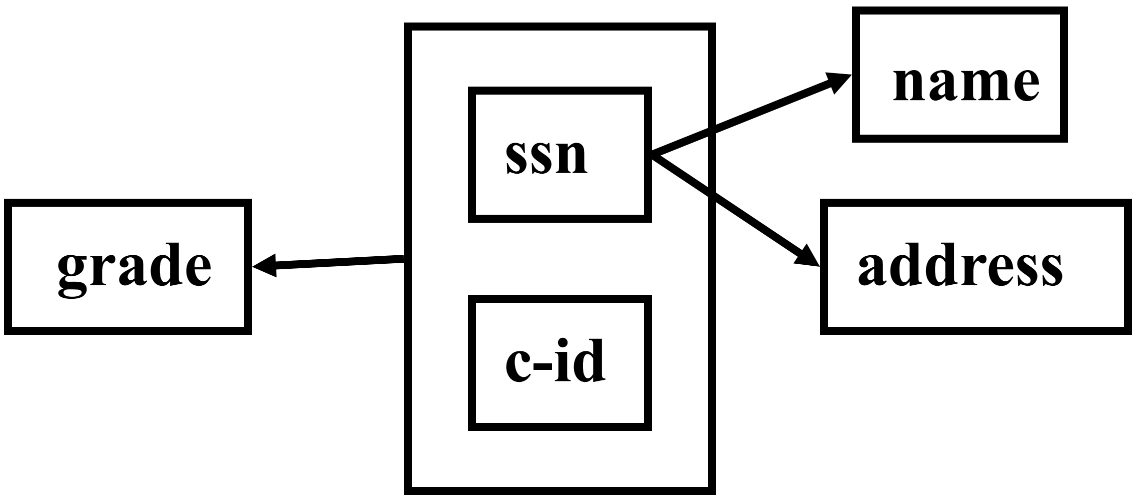
ssn->name, address

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

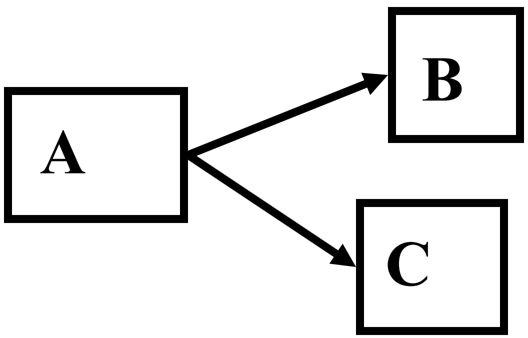
- pictorially: we want a 'star' shape



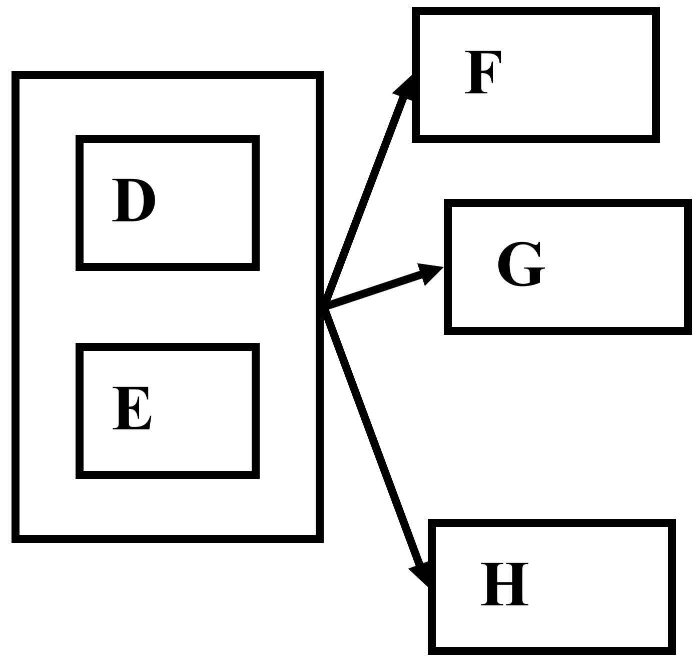
:**not** in BCNF

Normal forms - BCNF

- pictorially: we want a 'star' shape

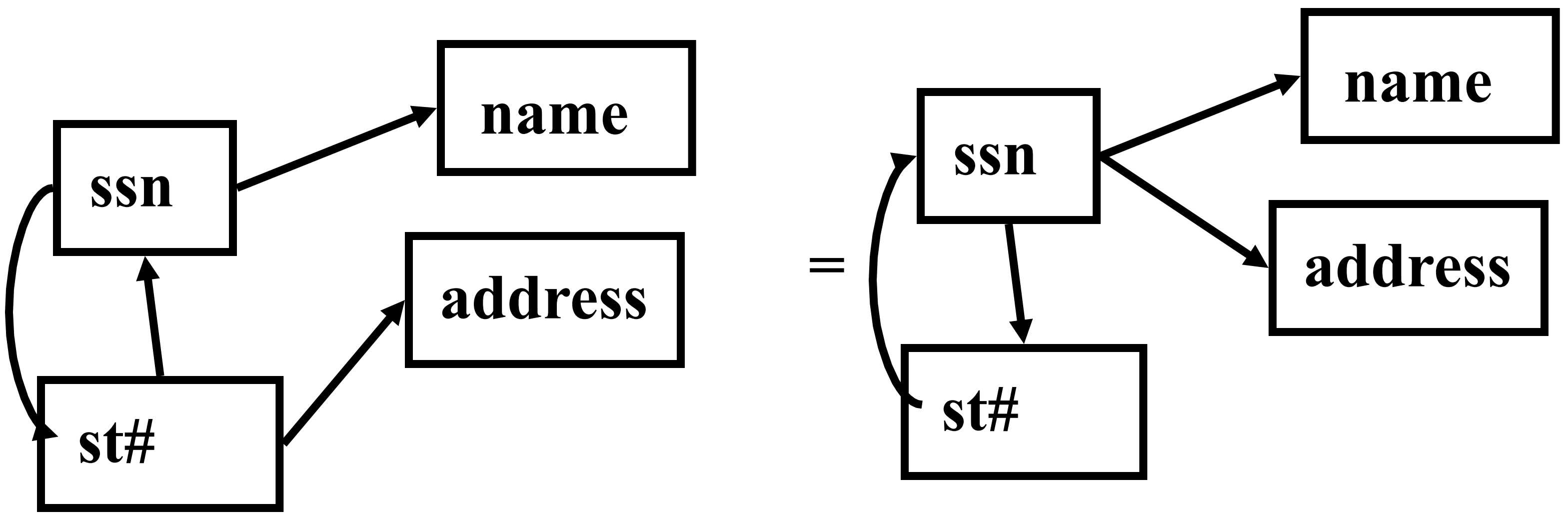


or



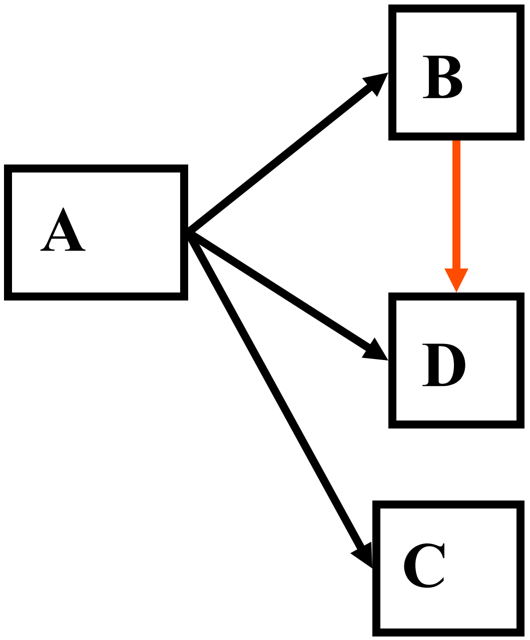
Normal forms - BCNF

- or a star-like: (eg., 2 cand. keys):
  - STUDENT(ssn, st#, name, address)

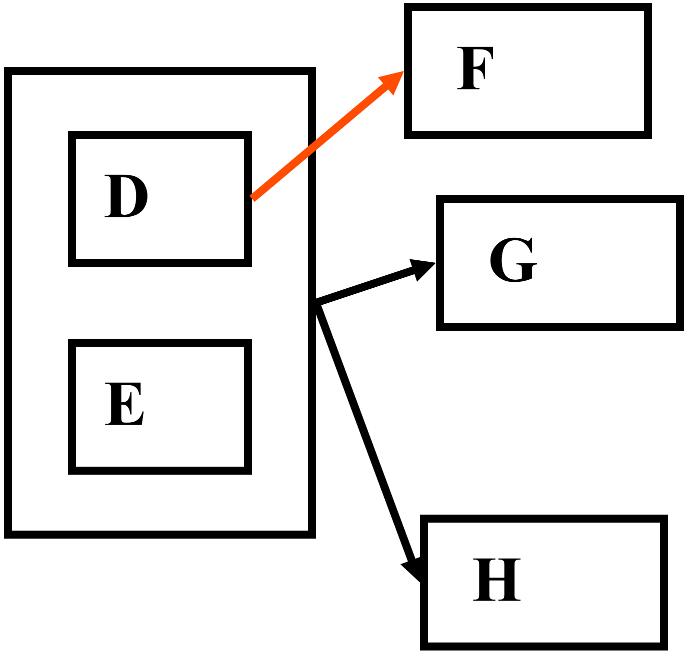


Normal forms - BCNF

- but not:



or



BCNF Decomposing Courses

- Schema is Courses(Number, DepartmentName, CourseName, Classroom, Enrollment, StudentName, Address)
- BCNF-violating FD is

Number DeparmentName → CourseName Classroom Enrollment

- Decompose Courses into

Courses1(Number, DepartmentName, CourseName, Classroom, Enrollment)

and

Courses2(Number, DepartmentName, StudentName, Address)

Are there any BCNF violations in the two new relations?

Another BCNF Example...

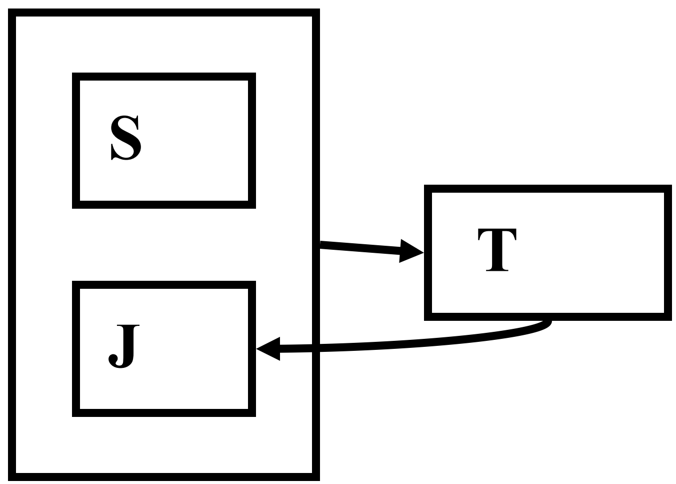
- Schema is Students(ID, Name, AdvisorId, AdvisorName, FavouriteAdvisorId)
- What are the FDs?
  - ID → Name FavouriteAdvisorId
  - AdvisorId → AdvisorName
- What is the key?
  - {ID, AdvisorId}
- Is there a BCNF violation?
  - Yes
- Let's use ID → Name FavouriteAdvisorId to decompose
- New relations?
  - Students1(ID, Name, FavouriteAdvisorId)
  - Students2(ID, AdvisorId, AdvisorName)

Another Example contd…

- What are the FDs in Student1(ID, Name, FavouriteAdvisorId)?
  - None that violate BCNF
- What are the FDs in Students2(ID, AdvisorID, AdvisorName)?
  - AdvisorID → AdvisorName
- Does it violate BCNF?
  - Yes!
- Rinse---Repeat the decomposition
- Let's use AdvisorID → AdvisorName for it
- New Relations:
  - Students2(ID, AdvisorId)
  - Students3(AdvisorId, AdvisorName)

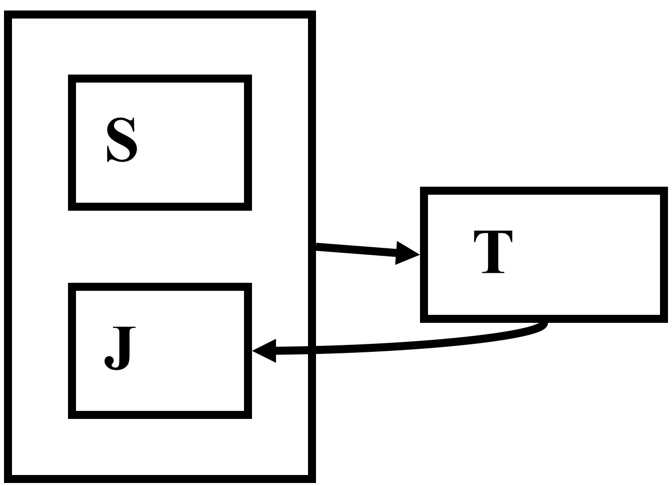
Normal forms - 3NF

- 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

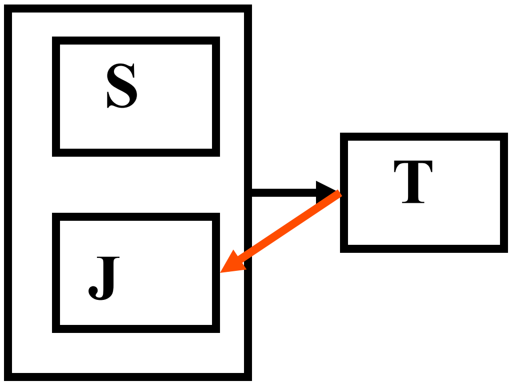
dependency preservation

Welcome 3NF!

(essentially define the issue away ☺)

Normal forms - 3NF

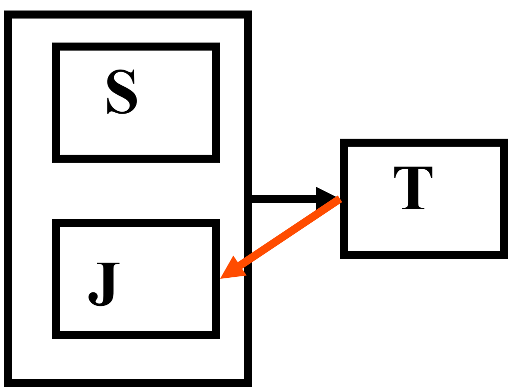
- STJ(Student, Teacher, subJect)
  - T-> J   S,J -> T



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

We prefer Synthesis as it is clearer and does not need ER diagrams

3NF Synthesis Algorithm: Details

Surprisingly Polynomial!

- Let F be the set of all FDs of R
- We will compute a lossless-join, dependency-preserving decomposition of R into S, where every relation in S is in 3NF

1. Find a canonical cover for F, say G
2. For every FD X → A in G, use X U A as the schema for one of the relations in S
3. If the attributes in none of the relations in S form a superkey for R, add another relation to S whose schema is a key for R (this will ensure that the decomp. is lossless)

3NF Synthesis Algorithm: Details

Correctness?
Tricky proof

- Let F be the set of all FDs of R
- We will compute a lossless-join, dependency-preserving decomposition of R into S, where every relation in S is in 3NF

1. Find a canonical cover for F, say G
2. For every FD X → A in G, use X U A as the schema for one of the relations in S
3. If the attributes in none of the relations in S form a superkey for R, add another relation to S whose schema is a key for R (this will ensure that the decomp. is lossless)

Normal forms - 3NF

Example:

R: ABC

F: A->B, C->B

- Q1: what is the cover?

- Q2: what is the decomposition to 3NF?

VirginiaTech

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: Step 1

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: one table each for the FDs

R1(A,B), R2(C,B), ...

But is it lossless?? Or equivalently do any of the relations in S form a superkey for R?

Normal forms - 3NF: Step 2

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)

(note that AC is a key for R)

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’)

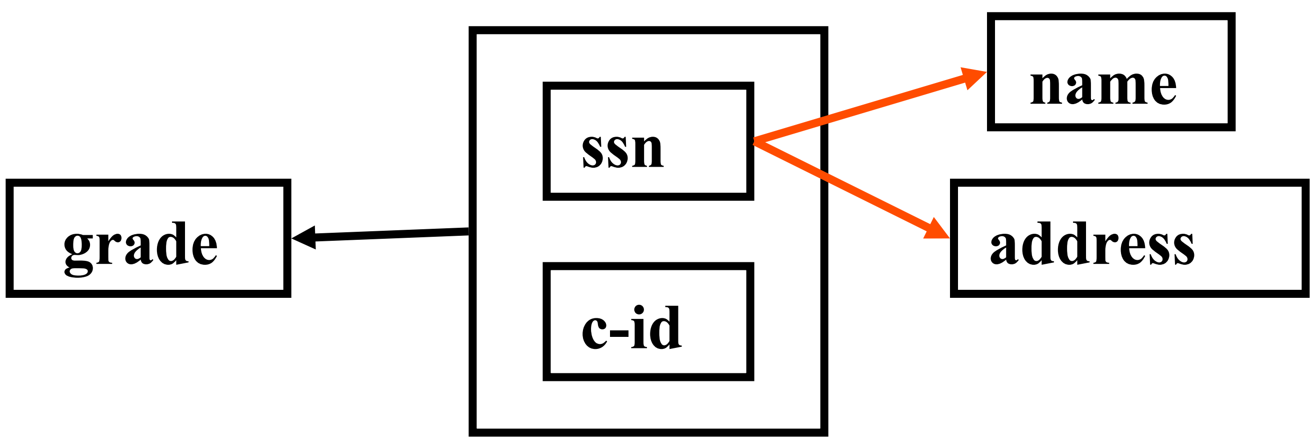
Ssn	Name	Dependents
123	Smith	Peter Mary John
234	Jones	Ann Michael

not 1NF

Normal forms - more details

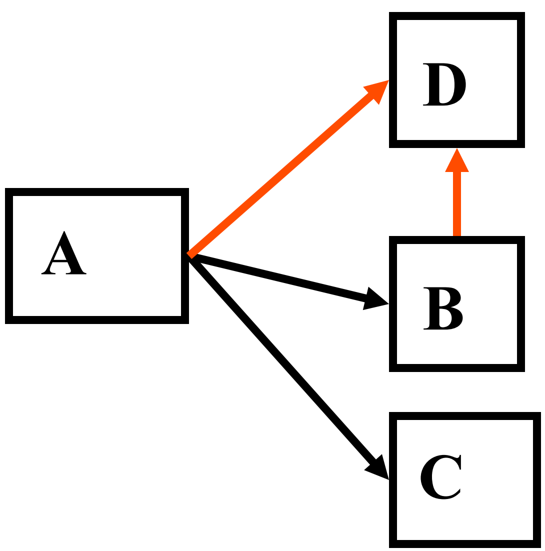
- 2NF: 1NF and non-key attr. fully depend on the key
- counter-example: TAKES1(ssn, c-id, grade, name, address)

- 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
- Fifth Normal Form: outside the scope of CS4604
- Sixth Normal Form: different versions exist. One version developed for temporal databases
- Seventh Normal Form
  - just kidding ☺

Normal forms - more details

- 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)

- Design Mantra:

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