Ask professor about his recommendation for data being repetitive in the model

Database Systems I

CMPT 354 Summer 2024

Zhengjie Miao

24 May 2024

Announcements (Wed., May 29)

- Homework 1 due tonight (11:59pm)
  - 2 maximum late days
- Homework 2 already assigned
  - Setup PostgreSQL now!
    - Will be used in future assignments

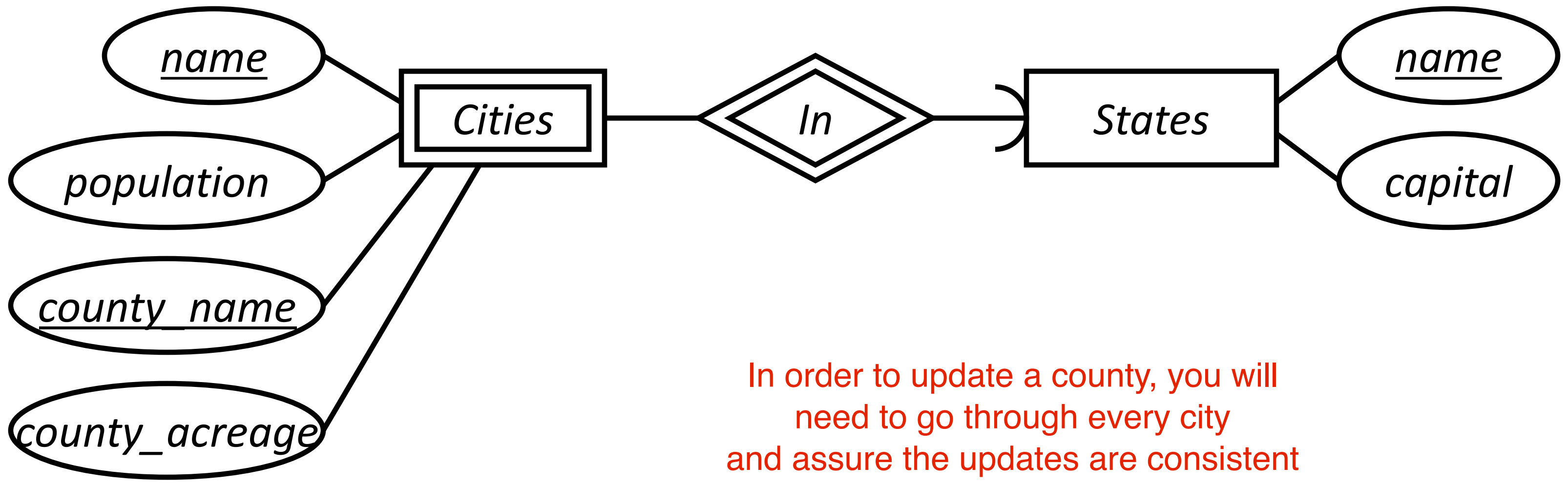
Database design steps

- Understand the real-world domain being modeled
- Specify it using a database design model (e.g., E/R)
    - Conceptual design
- Translate specification to the data model of DBMS (e.g., relational)
    - Logical design
- Schema refinement
- Create DBMS schema

Outline

- Database Design Theory
  - Functional Dependency
  - Schema Decomposition
  - Boyce-Codd Normal Form (BCNF) & BCNF Decomposition
  - Multivalued Dependency & 4th Normal Form

Recap: Case study 1 in E/R design



In order to update a county, you will
need to go through every city
and assure the updates are consistent

- County acreage information is repeated for every city in the county
  - ☞Redundancy is bad (why?)
- State capital should really be a city
  - ☞Should "reference" entities through explicit relationships

There are no other tables to store the user name, user id, and group id

Motivation

If we remove Milhouse, then we will have no record of him existing

If we want to update one the records here, you will need to make sure the update is proprogated to any other records with the old information

Certain constraints may need redundancy

uid	*uname*	*gid*
142	Bart	dps
123	Milhouse	gov
857	Lisa	abc
857	Lisa	gov
456	Ralph	abc
456	Ralph	gov
…	…	…

- Why is *UserGroup* (*uid*, *uname*, *gid*) a bad design?
  - It has redundancy — user name is recorded multiple times, once for each group that a user belongs to
    - Leads to update, insertion, deletion anomalies
- Wouldn't it be nice to have a systematic approach to detecting and removing redundancy in designs?
  - Dependencies, decompositions, and normal forms

Functional dependencies

- A functional dependency (FD) has the form $X \rightarrow Y$, where X and Y are sets of attributes in a relation R
  - X ***functionally determines Y*** if, for any tuples t_1 and t_2:
    - $t_1[A] = t_2[A]$ implies $t_1[B] = t_2[B]$
- $X \rightarrow Y$ means that whenever two tuples in R agree on all the attributes in X, they must also agree on all attributes in Y

X	Y	Z
a	b	c
a	b	?
…	…	…

Must be b

Could be anything

FD examples

Address (*street_address*, *city*, *state*, *zip*)

- *street_address*, *city*, *state* → *zip*
- *zip* → *city*, *state*
- *zip*, *state* → *zip*?
  - This is a trivial FD
  - Trivial FD: LHS ⊇ RHS
- *zip* → *state*, *zip*?
  - This is non-trivial, but not completely non-trivial
  - Completely non-trivial FD: LHS ∩ RHS = ∅

Redefining "keys" using FD's

A set of attributes K is a key for a relation R if

- $K \rightarrow$ all (other) attributes of R
  - That is, K is a "super key"

A key is minimal, but a superkey need not be minimal

- No proper subset of K satisfies the above condition
  - That is, K is minimal

Example

An FD holds, or does not hold on a table:

uid	*name*	*age*	*pop*
142	Bart	10	0.9
123	Milhouse	10	0.2
857	Lisa	8	0.7
459	Lisa	8	0.3

Ask professor if an FD holds for all possible relationship instances

√ uid → name

x name → pop

√ name,uid → age

Reasoning with FD's

Given a relation R and a set of FD's $\mathcal{F}$

- Does another FD follow from $\mathcal{F}$?
  - Are some of the FD's in $\mathcal{F}$ redundant (i.e., they follow from the others)?
- Is K a key of R?
  - What are all the keys of R?

Attribute closure

- Given R, a set of FD's $\mathcal{F}$ that hold in R, and a set of attributes Z in R:
  - The closure of Z (denoted Z^+) with respect to $\mathcal{F}$ is the set of all attributes $\{A_1, A_2, \dots\}$ functionally determined by Z (that is, $Z \rightarrow A_1 A_2 \dots$)
- Algorithm for computing the closure
  - Start with closure $= Z$
  - If $X \rightarrow Y$ is in $\mathcal{F}$ and X is already in the closure, then also add Y to the closure
  - Repeat until no new attributes can be added

A more complex example

Assuming a user can only join a group once

UserJoinsGroup (*uid*, *uname*, *twitterid*, *gid*, *fromDate*)

- Assume that there is a one-to-one correspondence between our users and Twitter accounts

- *uid* → *uname*, *twitterid*
- *twitterid* → *uid*
- *uid*, *gid* → *fromDate*

Not a good design, and we will see why shortly

Example of computing closure

$\mathcal{F}$ includes:

uid → *uname*, *twitterid*
twitterid → *uid*
uid, *gid* → *fromDate*

- $\{gid, twitterid\}^+$ = ?
- *twitterid* → *uid*
  - Add *uid*
  - Closure grows to { *gid*, *twitterid*, *uid* }
- *uid* → *uname*, *twitterid*
  - Add *uname*, *twitterid*
  - Closure grows to { *gid*, *twitterid*, *uid*, *uname* }
- *uid*, *gid* → *fromDate*
  - Add *fromDate*
  - Closure is now all attributes in *UserJoinsGroup*

Using attribute closure

Given a relation R and set of FD's $\mathcal{F}$

- Does another FD $X \to Y$ follow from $\mathcal{F}$?
  - Compute X^+ with respect to $\mathcal{F}$
  - If $Y \subseteq X^+$, then $X \to Y$ follows from $\mathcal{F}$
- Is K a key of R?
  - Compute K^+ with respect to $\mathcal{F}$
  - If K^+ contains all the attributes of R, K is a super key
  - Still need to verify that K is *minimal* (how?)
    - Hint: check the attribute closure of its proper subset.
    - i.e., Check that for no set X formed by removing attributes from K is K^+ the set of all attributes Ask professor to go over this

Rules of FD's

- Armstrong's axioms
  - Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$
    - uid, uname → uid Trivial dependency
  - Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
    - uid → *uname*
    - *uid, gid* → *uname, gid*
  - Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
    - Twitterid → *uid*
    - uid → *uname*
    - Twitterid → *uname*

Rules of FD's

- 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 $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
- Rules derived from axioms
  - Splitting: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
  - Combining: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- Using these rules, you can prove or disprove an FD given a set of FDs

Non-key FD's

- Consider a non-trivial FD $X \rightarrow Y$ where X is not a super key
  - Since X is not a super key, there are some attributes (say Z) that are not functionally determined by X

X	Y	Z
a	b	c_1
a	b	c_2
…	…	…

That a should be mapped to b is recorded multiple times:
redundancy, update/insertion/deletion anomaly

Example of redundancy

UserJoinsGroup (*uid*, *uname*, *twitterid*, *gid*, *fromDate*)

- *uid* → *uname*, *twitterid*

(… plus other FD's)

Issue with this table is that we have the users and the groups together.

uid	*uname*	*twitterid*	*gid*	*fromDate*
142	Bart	@BartJSimpson	dps	1987-04-19
123	Milhouse	@MilhouseVan_	gov	1989-12-17
857	Lisa	@lisasimpson	abc	1987-04-19
857	Lisa	@lisasimpson	gov	1988-09-01
456	Ralph	@ralphwiggum	abc	1991-04-25
456	Ralph	@ralphwiggum	gov	1992-09-01
…	…	…	…	…

Decomposition

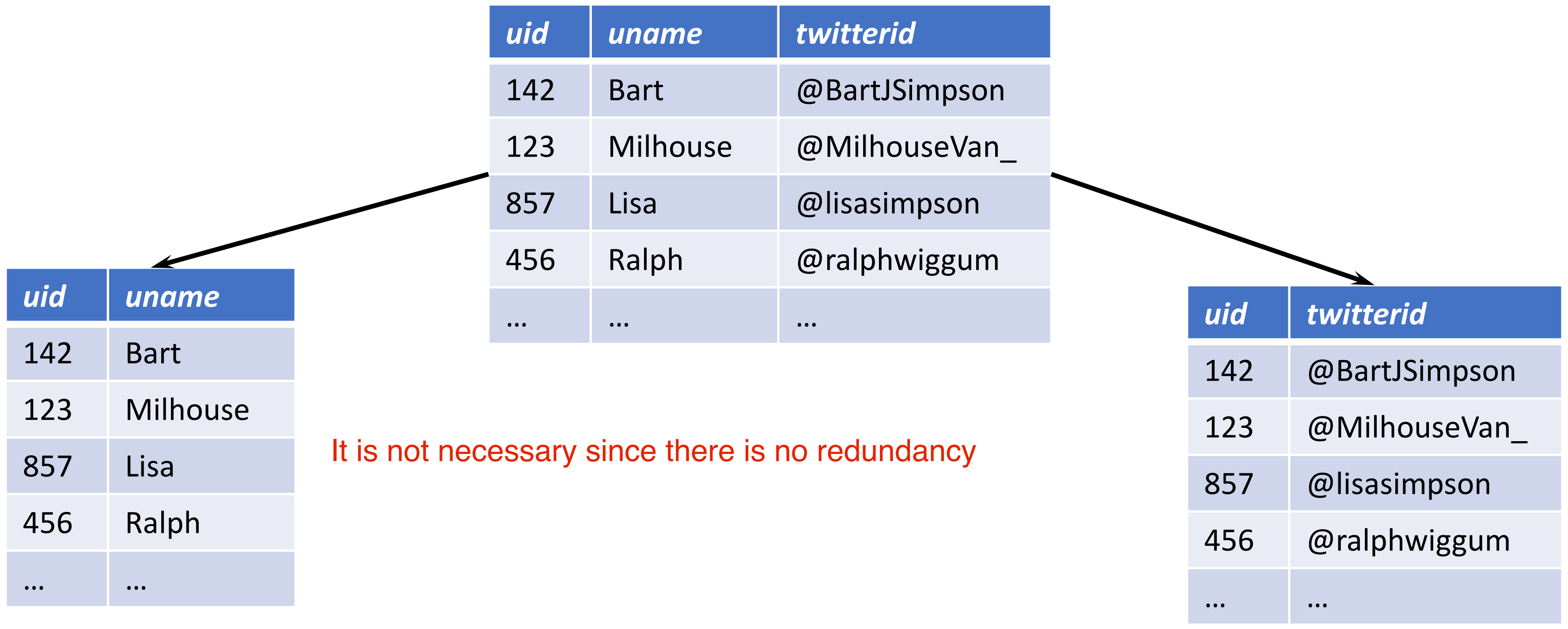
uid	*uname*	*twitterid*	*gid*	*fromDate*
142	Bart	@BartJSimpson	dps	1987-04-19
123	Milhouse	@MilhouseVan_	gov	1989-12-17
857	Lisa	@lisasimpson	abc	1987-04-19
857	Lisa	@lisasimpson	gov	1988-09-01
456	Ralph	@ralphwiggum	abc	1991-04-25
456	Ralph	@ralphwiggum	gov	1992-09-01
…	…	…	…	…

uid	*uname*	*twitterid*
142	Bart	@BartJSimpson
123	Milhouse	@MilhouseVan_
857	Lisa	@lisasimpson
456	Ralph	@ralphwiggum
…	…	…

uid	*gid*	*fromDate*
142	dps	1987-04-19
123	gov	1989-12-17
857	abc	1987-04-19
857	gov	1988-09-01
456	abc	1991-04-25
456	gov	1992-09-01
…	…	…

- Eliminates redundancy
- To get back to the original relation: $\bowtie$

Unnecessary decomposition



uid	uname	twitterid
142	Bart	@BartJSimpson
123	Milhouse	@MilhouseVan_
857	Lisa	@lisasimpson
456	Ralph	@ralphwiggum
…	…	…

uid	uname
142	Bart
123	Milhouse
857	Lisa
456	Ralph
…	…

It is not necessary since there is no redundancy

uid	twitterid
142	@BartJSimpson
123	@MilhouseVan_
857	@lisasimpson
456	@ralphwiggum
…	…

- Fine: join returns the original relation
- Unnecessary: no redundancy is removed; schema is more complicated

Bad decomposition

uid	*gid*	*fromDate*
142	dps	1987-04-19
123	gov	1989-12-17
857	abc	1987-04-19
857	gov	1988-09-01
456	abc	1991-04-25
456	gov	1992-09-01
…	…	…

What's wrong here?

uid	*gid*
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

uid	*fromDate*
142	1987-04-19
123	1989-12-17
857	1987-04-19
857	1988-09-01
456	1991-04-25
456	1992-09-01
…	…

You will get rows which never appeared in the original table

What will you get if you join the two decomposed tables?

- Association between *gid* and *fromDate* is lost
- Join returns more rows than the original relation

Lossless join decomposition

- Decompose relation R into relations S and T
  - $attrs(R) = attrs(S) \cup attrs(T)$
  - $S = \pi_{attrs(S)}(R)$
  - $T = \pi_{attrs(T)}(R)$
- The decomposition is a lossless join decomposition if, given known constraints such as FD's, we can guarantee that $R = S \bowtie T$

- Any decomposition gives $R \subseteq S \bowtie T$ (why?)
  - A lossy decomposition is one with $R \subset S \bowtie T$

Loss? But I got more rows!

- "Loss" refers not to the loss of tuples, but to the loss of information
  - Or, the ability to distinguish different original relations

uid	*gid*	*fromDate*
142	dps	1987-04-19
123	gov	1989-12-17
857	abc	1988-09-01
857	gov	1987-04-19
456	abc	1991-04-25
456	gov	1992-09-01
…	…	…

Swapping these two values gives another plausible relation;
no way to tell which one is the original!

uid	*gid*
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

uid	*fromDate*
142	1987-04-19
123	1989-12-17
857	1987-04-19
857	1988-09-01
456	1991-04-25
456	1992-09-01
…	…

Questions about decomposition

- When to decompose
- How to come up with a correct decomposition (i.e., lossless join decomposition)

An answer: BCNF

- A relation R is in Boyce-Codd Normal Form if
  - For every non-trivial FD $X \rightarrow Y$ in R, X is a super key
  - That is, all FDs follow from “key → other attributes”

- When to decompose
  - As long as some relation is not in BCNF
- How to come up with a correct decomposition
  - Always decompose on a BCNF violation (details next)

  ☞Then it is guaranteed to be a lossless join decomposition!

Example

Is this table in BCNF?

uid	*uname*	*twitterid*	*gid*	*fromDate*
142	Bart	@BartJSimpson	dps	1987-04-19
123	Milhouse	@MilhouseVan_	gov	1989-12-17
857	Lisa	@lisasimpson	abc	1987-04-19
857	Lisa	@lisasimpson	gov	1988-09-01
456	Ralph	@ralphwiggum	abc	1991-04-25
456	Ralph	@ralphwiggum	gov	1992-09-01
…	…	…	…	…

$\Longrightarrow$ **Not** in BCNF

FD1: uid → uname, twitterid
FD2: twitterid → uid
FD3: uid, gid → fromDate

Consider FD1: uid it is **not** a key

Since uid does not work as a key, you have a violation

What is the key?
{uid, gid}

BCNF decomposition algorithm

- Find a BCNF violation
    - That is, a non-trivial FD $X \rightarrow Y$ in R where X is not a super key of R
- Decompose R into R_1 and R_2, where
    - R_1 has attributes $X \cup Y$
    - R_2 has attributes $X \cup Z$, where Z contains all attributes of R that are in neither X nor Y
- Repeat until all relations are in BCNF

BCNF decomposition example

FD1: uid → uname, twitterid
FD2: twitterid → uid
FD3: uid, gid → fromDate

UserJoinsGroup (*uid, uname, twitterid, gid, fromDate*)

BCNF violation: *uid → uname, twitterid*

User (*uid, uname, twitterid*)

uid → uname, twitterid
twitterid → uid

BCNF

Not trivial since we have that uid determines another attribute?

Member (*uid, gid, fromDate*)

uid, gid → fromDate

BCNF

Another example

FD1: uid → uname, twitterid
FD2: twitterid → uid
FD3: uid, gid → fromDate

UserJoinsGroup (*uid, uname, twitterid, gid, fromDate*)

BCNF violation: *twitterid → uid*

UserId (*twitterid, uid*)

BCNF

UserJoinsGroup' (*twitterid, uname, gid, fromDate*)

twitterid → uname
twitterid, gid → fromDate

BCNF violation: *twitterid → uname*

BCNF violation since twitterid does not serve as a key

UserName (*twitterid, uname*)

BCNF

Member (*twitterid, gid, fromDate*)

BCNF

Why is BCNF decomposition lossless

Review this slide

Given non-trivial $X \rightarrow Y$ in R where X is not a super key of R, need to prove:

- Anything we project always comes back in the join:
$$R \subseteq \pi_{XY}(R) \bowtie \pi_{XZ}(R)$$
  - Sure; and it doesn't depend on the FD
- Anything that comes back in the join must be in the original relation:
$$R \supseteq \pi_{XY}(R) \bowtie \pi_{XZ}(R)$$
  - Proof will make use of the fact that $X \rightarrow Y$

BCNF = no redundancy?

- *User (uid, gid, place)*
  - A user can belong to multiple groups
  - A user can register places she’s visited
  - Groups and places have nothing to do with other
  - FD’s?
    - None
  - BCNF?
    - Yes
  - Redundancies?
    - Tons!

No dependencies since the keys are all the attributes?

We will always need to insert a copy of the group for each place they visit with the group

uid	*gid*	*place*
142	dps	Springfield
142	dps	Australia
456	abc	Springfield
456	abc	Morocco
456	gov	Springfield
456	gov	Morocco
…	…	…

Multivalued dependencies

- A multivalued dependency (MVD) has the form $X \twoheadrightarrow Y$, where X and Y are sets of attributes in a relation R
- $X \twoheadrightarrow Y$ means that whenever two rows in R agree on all the attributes of X, then we can swap their Y components and get two rows that are also in R

X	Y	Z
a	b_1	c_1
a	b_2	c_2
a	b_2	c_1
a	b_1	c_2
…	…	…

MVD examples

User (*uid*, *gid*, *place*)

- *uid* ↠ *gid*
- *uid* ↠ *place*
  - Intuition: given *uid*, *gid* and *place* are "independent"

Since the place a person visits is independent of the groups they are with, we can swap the places

- *uid*, *gid* ↠ *place*
  - Trivial: LHS ∪ RHS = all attributes of R
- *uid*, *gid* ↠ *uid*
  - Trivial: LHS ⊇ RHS

Complete MVD + FD rules

Review this slide

- FD reflexivity, augmentation, and transitivity
- MVD complementation:
  If $X \twoheadrightarrow Y$, then $X \twoheadrightarrow attrs(R) - X - Y$
- MVD augmentation:
  If $X \twoheadrightarrow Y$ and $V \subseteq W$, then $XW \twoheadrightarrow YV$
- MVD transitivity:
  If $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$, then $X \twoheadrightarrow Z - Y$
- Replication (FD is MVD):
  If $X \rightarrow Y$, then $X \twoheadrightarrow Y$
- Coalescence:
  If $X \twoheadrightarrow Y$ and $Z \subseteq Y$ and there is some W disjoint from Y such that $W \rightarrow Z$, then $X \rightarrow Z$

Try proving things using these!?

An elegant solution: chase

- Given a set of FD's and MVD's $\mathcal{D}$, does another dependency d (FD or MVD) follow from $\mathcal{D}$?
- Procedure
  - Start with the "if-part" of d, and treat them as "seed" tuples in a relation
  - Apply the given dependencies in $\mathcal{D}$ repeatedly
    - If we apply an FD, we infer equality of two symbols
    - If we apply an MVD, we infer more tuples
  - If we infer the "then-part" of d, we have a proof
  - Otherwise, if nothing more can be inferred, we have a counterexample

Proof by chase

- In $R(A, B, C, D)$, does $A \twoheadrightarrow B$ and $B \twoheadrightarrow C$ imply that $A \twoheadrightarrow C$?

Have:

	A	B	C	D
	a	b_1	c_1	d_1
	a	b_2	c_2	d_2
$A \twoheadrightarrow B$	a	b_2	c_1	d_1
	a	b_1	c_2	d_2
$B \twoheadrightarrow C$	a	b_2	c_1	d_2
	a	b_2	c_2	d_1
$B \twoheadrightarrow C$	a	b_1	c_2	d_1
	a	b_1	c_1	d_2

Need:

A	B	C	D	
a	b_1	c_2	d_1	✌
a	b_2	c_1	d_2	✌

Another proof by chase

- In $R(A, B, C, D)$, does $A \rightarrow B$ and $B \rightarrow C$ imply that $A \rightarrow C$?

Have:

A	B	C	D
a	b_1	c_1	d_1
a	b_2	c_2	d_2

Need:

$c_1 = c_2$ ✌

$A \rightarrow B$	$b_1 = b_2$
$B \rightarrow C$	$c_1 = c_2$

In general, with both MVD's and FD's,
chase can generate both new tuples and new equalities

Counterexample by chase

- In $R(A,B,C,D)$, does $A \twoheadrightarrow BC$ and $CD \to B$ imply that $A \to B$?

Have:

	A	B	C	D
	a	b_1	c_1	d_1
	a	b_2	c_2	d_2
$A \twoheadrightarrow BC$	a	b_2	c_2	d_1
$A \twoheadrightarrow BC$	a	b_1	c_1	d_2

Need: $b_1 = b_2$ 👎

Counterexample!

4NF

- A relation R is in Fourth Normal Form (4NF) if
  - For every non-trivial MVD $X \twoheadrightarrow Y$ in R, X is a superkey
  - That is, all FD's and MVD's follow from "key → other attributes" (i.e., no MVD's and no FD's besides key functional dependencies)

- 4NF is stronger than BCNF
  - Because every FD is also an MVD

4NF decomposition algorithm

- Find a 4NF violation
  - A non-trivial MVD $X \twoheadrightarrow Y$ in R where X is not a superkey
- Decompose R into R_1 and R_2, where
  - R_1 has attributes $X \cup Y$
  - R_2 has attributes $X \cup Z$ (where Z contains R attributes not in X or Y)
- Repeat until all relations are in 4NF

- Almost identical to BCNF decomposition algorithm
- Any decomposition on a 4NF violation is lossless

4NF decomposition example

uid	*gid*	*place*
142	dps	Springfield
142	dps	Australia
456	abc	Springfield
456	abc	Morocco
456	gov	Springfield
456	gov	Morocco
…	…	…

$$\begin{tikzcd}
& \textit{User}\ (\textit{uid}, \textit{gid}, \textit{place}) \arrow[dl] \arrow[dr] & \\
\textit{Member}\ (\textit{uid}, \textit{gid}) & & \textit{Visited}\ (\textit{uid}, \textit{place})
\end{tikzcd}$$

4NF violation: *uid* $\twoheadrightarrow$ *gid*

Member (*uid*, *gid*)

4NF

uid	*gid*
142	dps
456	abc
456	gov
…	…

Visited (*uid*, *place*)

4NF

uid	*place*
142	Springfield
142	Australia
456	Springfield
456	Morocco
…	…

Third Normal Form (3NF)

- R with FDs *F* is in **3NF** if, for all $X \rightarrow Y$ in F^+
  - $Y \in X$ (called a *trivial* FD), or
  - **X** is a superkey of R, **or**
  - **Y** is part of some **candidate** key (not superkey!) for R

- If R is in BCNF, obviously in 3NF.
- If R is in 3NF, some redundancy is possible.
- It is a compromise, used when BCNF not achievable
  - (e.g., no "good" decomp, or performance considerations).
  - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

Summary

- Philosophy behind BCNF, 4NF:
  Data should depend on the key,
  the whole key,
  and nothing but the key!
  - You could have multiple keys though
- Other normal forms
  - 3NF: More relaxed than BCNF; will not remove redundancy if doing so makes FDs harder to enforce
  - 2NF: Slightly more relaxed than 3NF
  - 1NF: All column values must be atomic