

Relational Database Design Theory

Introduction to Databases

CompSci 316 Fall 2020



DUKE
COMPUTER SCIENCE

Announcements (Tue. Sept 22)

- HW4: due tomorrow (Wed)
- Midterm next Tuesday 09/29
 - See gradescope for policy

Today's plan

- Start database design theory
 - Functional dependency, BCNF
- Review some concepts in between and at the end
 - Weak entity set, ISA, multiplicity, etc. in ER diagram
 - Outer joins, different join types
 - Triggers
 - EXISTS
 - Foreign keys

Motivation



<i>uid</i>	<i>uname</i>	<i>gid</i>
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
- $X \rightarrow Y$ 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* \rightarrow *zip*
- *zip* \rightarrow *city*, *state*
- *zip*, *state* \rightarrow *zip*?
 - This is a trivial FD
 - **Trivial FD**: LHS \supseteq RHS
- *zip* \rightarrow *state*, *zip*?
 - This is non-trivial, but not completely non-trivial
 - **Completely non-trivial FD**: LHS \cap RHS = \emptyset

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

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

Example
On board
Using next slide

A more complex example

UserJoinsGroup (uid, unname, twitterid, gid, fromDate)

Assume that there is a 1-1 correspondence between our users and Twitter accounts

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

Not a good design, and we will see why shortly

Example of computing closure

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

\mathcal{F} includes:

$uid \rightarrow uname, twitterid$

$twitterid \rightarrow uid$

$uid, gid \rightarrow fromDate$

Using attribute closure

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

- Does another FD $X \rightarrow Y$ follow from \mathcal{F} ?
 - Compute X^+ with respect to \mathcal{F}
 - If $Y \subseteq X^+$, then $X \rightarrow Y$ 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?)

We already used these intuitive rules but check yourself again!

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

Announcements (Thu. Sept 24)

- MS1 due tonight 09/24 on gradescope!
- Midterm next Tuesday 09/29
 - Gradescope assignment for policy due tonight 09/24
 - See sakai/piazza for more details
 - No makeup exam
 - No extra time after 2 hours – it is 1:15 hours for exam + 45 mins as backup
 - Make sure to follow all policy
 - Everything until today included (practice problems will be posted)
- Sample midterm on Sakai
- Sudeepa's OH: Fri (tomorrow, 09/25) – 10-11 ET and Mon (09/28) – 11-12 ET
- STINF for assignments: 2 extra days, after that Dean's excuse needed

(Problems with) 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 b is associated with a 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)

<i>uid</i>	<i>uname</i>	<i>twitterid</i>	<i>gid</i>	<i>fromDate</i>
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
...

What are the problems? How do we fix them?

Decomposition

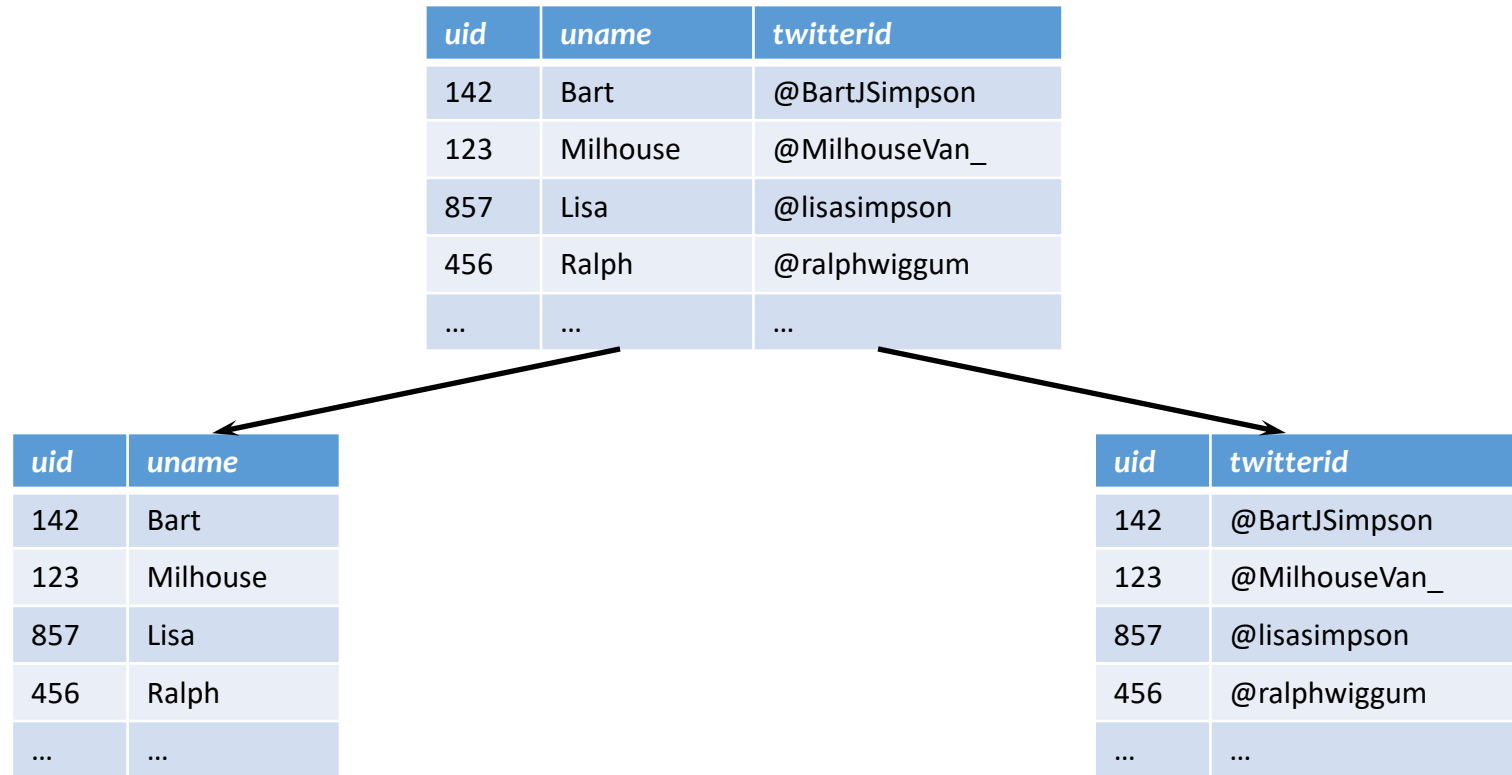
<i>uid</i>	<i>uname</i>	<i>twitterid</i>	<i>gid</i>	<i>fromDate</i>
142	Bart	@BartJSimpson	dps	1987-04-19
123	Milhouse	@MilhouseVan_	gov	1989-12-17
857	Lisa	@lisasimpson	abc	1987-04-19
857	Lisa	@lisasimpson	gov	1988-09-01
456	Ralph	@ralphwiggum	abc	1991-04-25
456	Ralph	@ralphwiggum	gov	1992-09-01
...

<i>uid</i>	<i>uname</i>	<i>twitterid</i>
142	Bart	@BartJSimpson
123	Milhouse	@MilhouseVan_
857	Lisa	@lisasimpson
456	Ralph	@ralphwiggum
...

<i>uid</i>	<i>gid</i>	<i>fromDate</i>
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
...

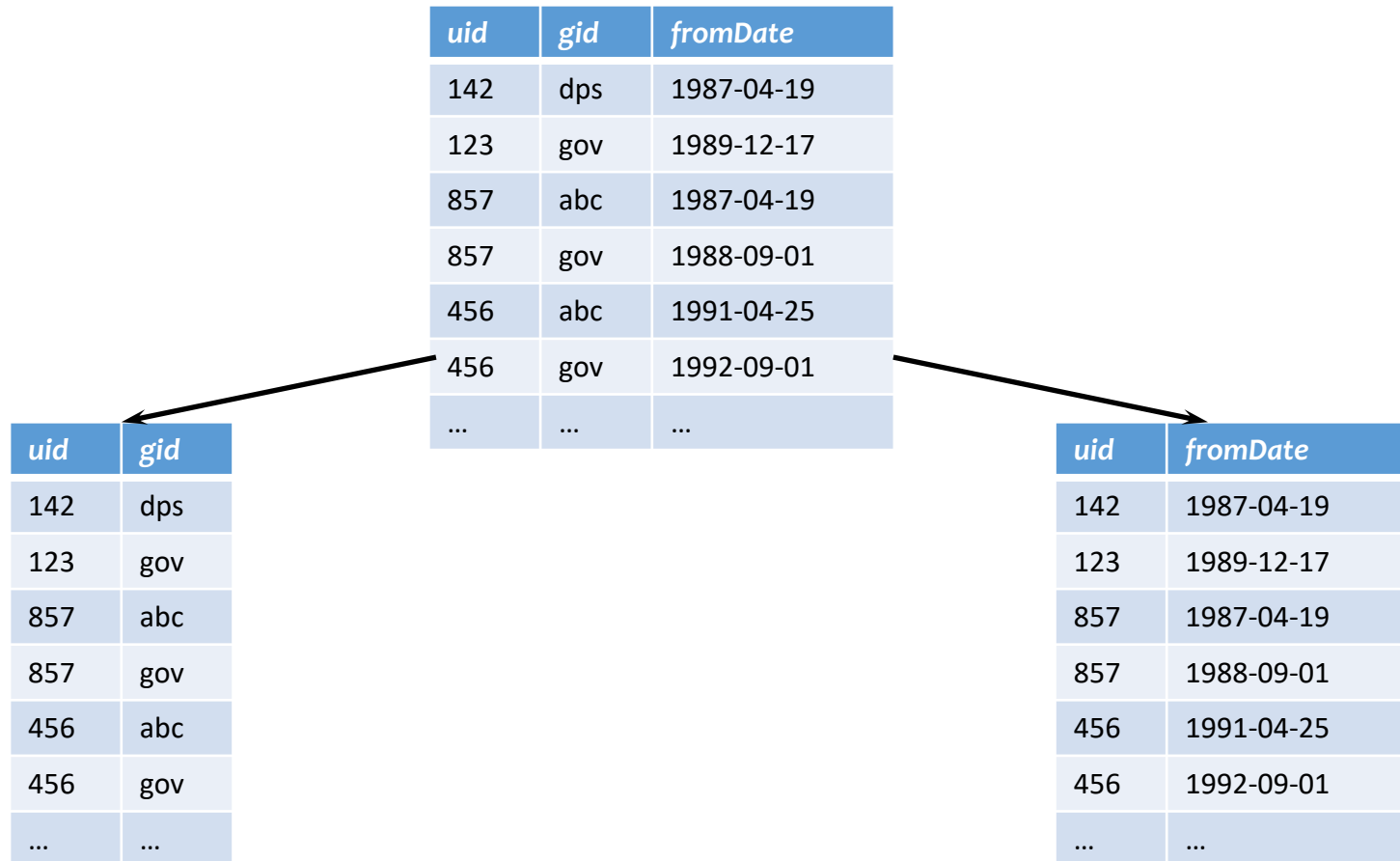
- Eliminates redundancy
- To get back to the original relation: ⋈

Unnecessary decomposition



- Fine: join returns the original relation
- Unnecessary: no redundancy is removed; schema is more complicated (and *uid* is stored twice!)

Bad decomposition



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

Lossless join decomposition

Example on board
Check definition yourself

- 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!

End of Lecture 09/24
Midterm syllabus
up to here

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

<i>uid</i>	<i>gid</i>	<i>fromDate</i>
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
...

No way to tell
which is the original relation

<i>uid</i>	<i>gid</i>
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
...	...

<i>uid</i>	<i>fromDate</i>
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 \rightarrow 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!

BCNF decomposition algorithm

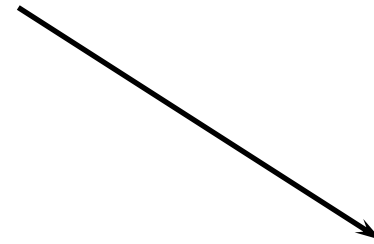
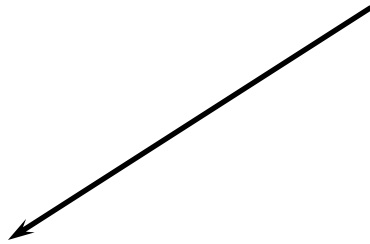
- Find a **BCNF violation**
 - That is, a non-trivial FD $X \rightarrow Y$ in R where X is **not** a super key of R
- Decompose R into R_1 and R_2 , where
 - R_1 has attributes $X \cup Y$
 - R_2 has attributes $X \cup Z$, where Z contains all attributes of R that are in neither X nor Y
- Repeat until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

BCNF decomposition example

$uid \rightarrow uname, twitterid$
 $twitterid \rightarrow uid$
 $uid, gid \rightarrow fromDate$

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

BCNF violation: $uid \rightarrow uname, twitterid$



User ($uid, uname, twitterid$)

$uid \rightarrow uname, twitterid$
 $twitterid \rightarrow uid$

BCNF

Member ($uid, gid, fromDate$)

$uid, gid \rightarrow fromDate$

BCNF

Another example

$uid \rightarrow uname, twitterid$
 $twitterid \rightarrow uid$
 $uid, gid \rightarrow fromDate$

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

BCNF violation: $twitterid \rightarrow uid$

UserId ($twitterid, uid$)

BCNF

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

$twitterid \rightarrow uname$
 $twitterid, gid \rightarrow fromDate$

BCNF violation: $twitterid \rightarrow uname$

UserName ($twitterid, uname$) Member ($twitterid, gid, fromDate$)

BCNF

BCNF

Why is BCNF decomposition lossless

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
- Check and prove yourself!
- 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 $X \rightarrow Y$

Recap

- Functional dependencies: a generalization of the key concept
- Non-key functional dependencies: a source of redundancy
- BCNF decomposition: a method for removing redundancies
 - BCNF decomposition is a lossless join decomposition
- BCNF: schema in this normal form has no redundancy due to FD's

Summary

- Philosophy behind BCNF:
Data should depend on the key,
the whole key,
and nothing but the key!
 - You could have multiple keys though
- Other normal forms
 - 4NF and Multi-valued-dependencies : later in the course
 - Not covered
 - 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

