CompSci 516
Database Systems

Lecture 5
Design Theory and Normalization

Instructor: Sudeepa Roy

Where are we now?

We learnt
- ✓ Relational Model and Query Languages
  - ✓ SQL, RA, RC
  - ✓ Postgres (DBMS)
  - ✓ XML (overview)
  - HW1

Next
- Database Normalization
  - (for good schema design)

Design Theory and Normalization

Reading Material

- Database normalization
  - [RG] Chapter 19.1 to 19.5, 19.6.1, 19.8 (overview)
  - [GUW] Chapter 3

Acknowledgement:
- The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.
- Some slides have been adapted from slides by Profs. Magda Balazinska, Dan Suciu, and Jun Yang

What will we learn?

- What goes wrong if we have redundant info in a database?
- Why and how should you refine a schema?
- Functional Dependencies – a new kind of integrity constraints (IC)
- Normal Forms
- How to obtain those normal forms

Example

The list of hourly employees in an organization

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

- key = SSN

Example

The list of hourly employees in an organization

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

- key = SSN
- Suppose for a given rating, there is only one hourly_wage value
- Redundancy in the table
- Why is redundancy bad?

Why is redundancy bad?

The list of hourly employees in an organization

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

Why is redundancy bad?

The list of hourly employees in an organization

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10 → 9	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

Why is redundancy bad?

The list of hourly employees in an organization

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

Why is redundancy bad?

The list of hourly employees in an organization

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

Nulls may or may not help

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

Summary: Redundancy

- Solution?
  - decomposition of schema

Decomposition

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

ssn (S)	name (N)	lot (L)	rating (R)	hours-worked (H)
111-11-1111	Attishoo	48	8	40
222-22-2222	Smiley	22	8	30
333-33-3333	Smethurst	35	5	30
444-44-4444	Guldu	35	5	32
555-55-5555	Madayan	35	8	40

rating	hourly_wage
8	10
5	7

Decompositions should be used judiciously

1. Do we need to decompose a relation?
   - Several normal forms
   - If a relation is not in one of them, may need to decompose further
2. What are the problems with decomposition?
   - Lossless joins (soon)
   - Performance issues -- decomposition may both
     - help performance (for updates, some queries accessing part of data), or
     - hurt performance (new joins may be needed for some queries)

Functional Dependencies (FDs)

- A functional dependency (FD) X → Y holds over relation R if, for every allowable instance *r* of R:
  - i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree
  - X and Y are *sets* of attributes
  - $t1 \in r,\ t2 \in r,\ \Pi_X(t1) = \Pi_X(t2)$ implies $\Pi_Y(t1) = \Pi_Y(t2)$

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

What is an FD here?

Functional Dependencies (FDs)

- A functional dependency (FD) X → Y holds over relation R if, for every allowable instance *r* of R:
  - i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree
  - X and Y are *sets* of attributes
  - $t1 \in r,\ t2 \in r,\ \Pi_X(t1) = \Pi_X(t2)$ implies $\Pi_Y(t1) = \Pi_Y(t2)$

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

What is an FD here?

AB → C

Note that, AB is not a key

not a correct question though.. see next slide!

Functional Dependencies (FDs)

- An FD is a statement about all allowable relations
  - Must be identified based on semantics of application
  - Given some allowable instance *r1* of R, we can check if it violates some FD *f*, but we cannot tell if *f* holds over R
- K is a candidate key for R means that K →R
  - denoting R = all attributes of R too
  - However, S →R does not require S to be minimal
  - e.g. S can be a superkey

Example

- Consider relation obtained from Hourly_Emps:
  - Hourly_Emps (ssn, name, lot, rating, hourly_wage, hours_worked)
- Notation: We will denote a relation schema by listing the attributes: SNLRWH
  - Basically the set of attributes {S,N,L,R,W,H}
  - here first letter of each attribute
- FDs on Hourly_Emps:
  - ssn is the key: S → SNLRWH
  - rating determines hourly_wages: R → W

Armstrong's Axioms

- X, Y, Z are sets of attributes
- Reflexivity: If X ⊇ Y, then X → Y
- Augmentation: If X → Y, then XZ → YZ for any Z
- Transitivity: If X → Y and Y → Z, then X → Z

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

Apply these rules on AB → C and check

Armstrong's Axioms

- X, Y, Z are sets of attributes
- Reflexivity: If X ⊇ Y, then X → Y
- Augmentation: If X → Y, then XZ → YZ for any Z
- Transitivity: If X → Y and Y → Z, then X → Z
- These are sound and complete inference rules for FDs
  - sound: then only generate FDs in F^+ for F
  - complete: by repeated application of these rules, all FDs in F^+ will be generated

Additional Rules

- Follow from Armstrong's Axioms
- Union: If X → Y and X → Z, then X → YZ
- Decomposition: If X → YZ, then X → Y and X → Z

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a2	b2	c2	d1
a2	b2	c2	d2

A → B, A → C
A → BC

A → BC
A → B, A → C

Closure of a set of FDs

- Given some FDs, we can usually infer additional FDs:
  - SSN → DEPT, and DEPT → LOT implies SSN → LOT
- An FD *f* is implied by a set of FDs *F* if *f* holds whenever all FDs in *F* hold.
- F^+
  = closure of F is the set of all FDs that are implied by *F*

To check if an FD belongs to a closure

- Computing the closure of a set of FDs can be expensive
  - Size of closure can be exponential in #attributes
- Typically, we just want to check if a given FD *X → Y* is in the closure of a set of FDs *F*
- No need to compute F^+

1. Compute attribute closure of X (denoted X^+) wrt *F:*
   - Set of all attributes A such that X → A is in F^+
2. Check if Y is in X^+

Computing Attribute Closure

Algorithm:

- closure = X
- Repeat until no change
  - if there is an FD $U \rightarrow V$ in F such that $U \subseteq$ closure, then closure = closure $\cup$ V
- Does F = {$A \rightarrow B$, $B \rightarrow C$, $C\,D \rightarrow E$ } imply $A \rightarrow E$?
  - i.e, is $A \rightarrow E$ in the closure F^+? Equivalently, is E in A^+?

Normal Forms

- Question: given a schema, how to decide whether any schema refinement is needed at all?
- If a relation is in a certain normal forms, it is known that certain kinds of problems are avoided/minimized
- Helps us decide whether decomposing the relation is something we want to do

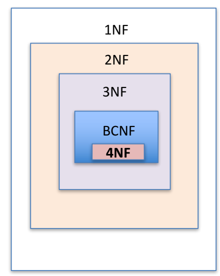
FDs play a role in detecting redundancy

Example

- Consider a relation R with 3 attributes, ABC
  - No FDs hold: There is no redundancy here – no decomposition needed
  - Given $A \rightarrow B$: Several tuples could have the same A value, and if so, they'll all have the same B value – redundancy – decomposition may be needed if A is not a key
- Intuitive idea:
  - if there is any non-key dependency, e.g. $A \rightarrow B$, decompose!

Normal Forms

R is in 4NF
$\Rightarrow$ R is in BCNF
$\Rightarrow$ R is in 3NF
$\Rightarrow$ R is in 2NF (a historical one)
$\Rightarrow$ R is in 1NF (every field has atomic values)



Only BCNF and 4NF are covered in the class

Boyce-Codd Normal Form (BCNF)

- Relation R with FDs *F* is in BCNF if, for all $X \rightarrow A$ in F
  - $A \in X$ (called a trivial FD), or
  - X contains a key for R
    - i.e. X is a superkey

Next lecture: BCNF decomposition algorithm

Decomposition

(on twitter)

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
…	…	…	…	…

- User id
- user name
- Twitter id
- Group id
- Joining Date (to a group)

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
…	…

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 (and *uid* is stored twice!)

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
…	…	…

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
…	…

- 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 $R = S \bowtie T$
- $R \subseteq S \bowtie T$ or $R \supseteq S \bowtie T$?
- 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
…	…	…

No way to tell which is the original relation

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
…	…

BCNF decomposition algorithm

- Find a BCNF violation
  - That is, a non-trivial FD $X \to Y$ 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
- Also gives a lossless decomposition!

BCNF decomposition example - 1

- CSJDPQV, key C, F = {JP → C, SD → P, J → S}
  - To deal with SD → P, decompose into SDP, CSJDQV.
  - To deal with J → S, decompose CSJDQV into JS and CJDQV
- Is JP → C a violation of BCNF?
- Note:
  - several dependencies may cause violation of BCNF
  - The order in which we pick them may lead to very different sets of relations
  - there may be multiple correct decompositions (can pick J → S first)

BCNF decomposition example - 2

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

UserJoinsGroup (*uid, uname, twitterid, gid, fromDate*)
BCNF violation: *uid → uname, twitterid*

User (*uid, uname, twitterid*)
uid → uname, twitterid
twitterid → uid
BCNF

Member (*uid, gid, fromDate*)
uid, gid → fromDate
BCNF

BCNF decomposition example - 3

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

UserJoinsGroup (*uid, uname, twitterid, gid, fromDate*)
BCNF violation: *twitterid → uid*

apply Armstrong's axioms and rules!

UserId (*twitterid, uid*)
BCNF

UserJoinsGroup' (*twitterid, uname, gid, fromDate*)
twitterid → uname
twitterid, gid → fromDate
BCNF violation: *twitterid → uname*

UserName (*twitterid, uname*)
BCNF

Member (*twitterid, gid, fromDate*)
BCNF

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

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!

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

Verify these yourself!

Complete MVD + FD rules

- 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!?

Read this slide after looking at the examples

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 premise 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 conclusion 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: $c_1 = c_2$

$A \rightarrow B$ $\quad b_1 = b_2$

$B \rightarrow C$ $\quad c_1 = c_2$

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 $CD \rightarrow B$ imply that $A \rightarrow B$?

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	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 $\rightarrow$ 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 a 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

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

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

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

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
…	…

Other kinds of dependencies and normal forms

- Dependency preserving decompositions
- Join dependencies
- Inclusion dependencies
- 5NF, 3NF, 2NF
- See book if interested (not covered in class)

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
- Redundancy is not desired typically
  - not always, mainly due to performance reasons
- Functional/multivalued dependencies – capture redundancy
- Decompositions – eliminate dependencies
- Normal forms
  - Guarantees certain non-redundancy
  - BCNF, and 4NF
- Lossless join
- How to decompose into BCNF, 4NF
- Chase