Database Systems
Lecture #5 BCNF&3NF

Guifeng Zheng
SYSU

Agenda

- Last time: FDs
- This time:

1. Anomalies
2. Normalization: BCNF & 3NF

- Next time: RA & SQL

Types of anomalies

- Redundancy
  - Repeat info unnecessarily in several tuples
- Update anomalies:
  - Change info in one tuple but not in another
- Deletion anomalies:
  - Delete some values & lose other values too
- Insert anomalies:
  - Inserting row means having to insert other, separate info / null-ing it out

Examples of anomalies

Name	<u>SSN</u>	Mailing-address	<u>Phone</u>
Michael	123	NY	212-111-1111
Michael	123	NY	917-111-1111
Hilary	456	DC	202-222-2222
Hilary	456	DC	914-222-2222
Bill	789	Chappaqua	914-222-2222
Bill	789	Chappaqua	212-333-3333

SSN → Name, Mailing-address

SSN $\nrightarrow$ Phone

- Redundancy: name, maddress
- Update anomaly: Bill moves
- Delete anom.: Bill doesn't pay bills, lose phones → lose Bill!
- Insert anom: can't insert someone without a (non-null) phone
- Underlying cause: SSN-phone is many-many
- Effect: *partial dependency* ssn → name, maddress,
  - Whereas key = {ssn,phone}

Decomposition by projection

- Soln: replace anomalous R with *projections* of R onto two subsets of attributes
- **Projection:** an operation in Relational Algebra
  - Corresponds to *SELECT* command in SQL

- Projecting R onto attributes ($A_1,\ldots,A_n$) means *removing* all other attributes
  - Result of projection is *another relation*
  - Yields tuples whose fields are $A_1,\ldots,A_n$
  - Resulting duplicates ignored

Projection for decomposition

$R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p)$

$R_1(A_1, \ldots, A_n, B_1, \ldots, B_m)$

$R_2(A_1, \ldots, A_n, C_1, \ldots, C_p)$

R_1 = projection of R on $A_1, \ldots, A_n, B_1, \ldots, B_m$

R_2 = projection of R on $A_1, \ldots, A_n, C_1, \ldots, C_p$

$A_1, \ldots, A_n \cup B_1, \ldots, B_m \cup C_1, \ldots, C_p$ = all attributes

R_1 and R_2 may (/not) be reassembled to produce original R

Decomposition example

Break the relation into two:

Name	SSN	Mailing-address	Phone
Michael	123	NY	212-111-1111
Michael	123	NY	917-111-1111
Hilary	456	DC	202-222-2222
Hilary	456	DC	914-222-2222
Bill	789	Chappaqua	914-222-2222
Bill	789	Chappaqua	212-333-3333

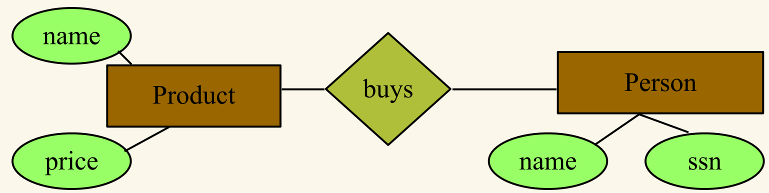
Name	SSN	Mailing-address
Michael	123	NY
Hilary	456	DC
Bill	789	Chappaqua

SSN	Phone
123	212-111-1111
123	917-111-1111
456	202-222-2222
456	914-222-2222
789	914-222-2222
789	212-333-3333

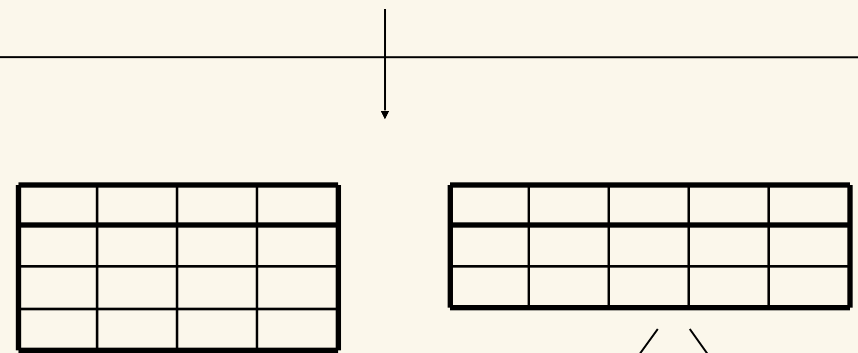
- The anomalies are gone
  - No more redundant data
  - Easy to for Bill to move
  - Okay for Bill to lose all phones

Thus: high-level strategy

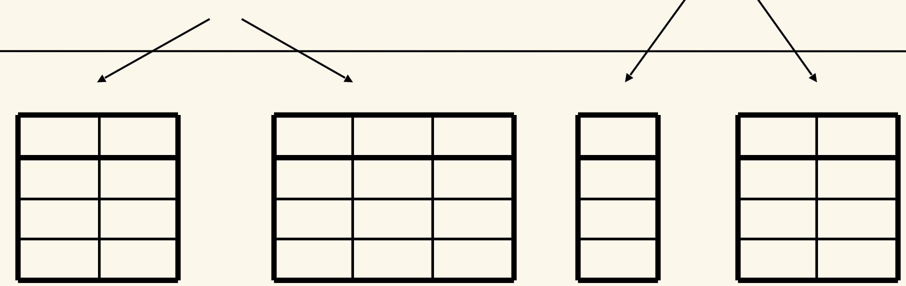
E/R Model:



Relational Model:
plus FD's



Normalization:
Eliminates anomalies



Using FDs to produce good schemas

1. Start with set of relations
2. Define FDs (and keys) for them based on real world
3. Transform your relations to “normal form” (normalize them)
   - Do this using “decomposition”

- Intuitively, good design means
  - No anomalies
  - Can reconstruct all (and only the) original information

Decomposition terminology

- **Projection**: eliminating certain atts from relation
- **Decomposition**: separating a relation into two by projection
- **Join**: (re)assembling two relations
  - Whenever a row from R_1 and a row from R_2 have the same value for some atts *A,* join together to form a row of R_3
- If exactly the original rows are reproduced by joining the relations, then the decomposition was **lossless**
  - We join on the attributes R_1 and R_2 have in common (As)
- If it can't, the decomposition was **lossy**

Lossless Decompositions

A decomposition is *lossless* if we can recover:

R($\underline{A}$,B,C)

Decompose

R1($\underline{A}$,B) R2($\underline{A}$,$\underline{C}$)

Recover

R'(A,B,C) should be the same as R(A,B,C)

R' is in general larger than R. Must ensure R' = R

Lossless decomposition

- Sometimes the same set of data is reproduced:

Name	Price	Category
Word	100	WP
Oracle	1000	DB
Access	100	DB

Name	Price
Word	100
Oracle	1000
Access	100

Name	Category
Word	WP
Oracle	DB
Access	DB

- (Word, 100) + (Word, WP) → (Word, 100, WP)
- (Oracle, 1000) + (Oracle, DB) → (Oracle, 1000, DB)
- (Access, 100) + (Access, DB) → (Access, 100, DB)

Lossy decomposition

- Sometimes it's not:

Name	Price	Category
Word	100	WP
Oracle	1000	DB
Access	100	DB

What's wrong?

Category	Name
WP	Word
DB	Oracle
DB	Access

Category	Price
WP	100
DB	1000
DB	100

- (Word, WP) + (100, WP) → (Word, 100, WP)
- (Oracle, DB) + (1000, DB) → (Oracle, 1000, DB)
- (Oracle, DB) + (100, DB) → (Oracle, **100**, DB)
- (Access, DB) + (1000, DB) → (Access, **1000**, DB)
- (Access, DB) + (100, DB) → (Access, 100, DB)

Ensuring lossless decomposition

$R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p)$

$R_1(A_1, \ldots, A_n, B_1, \ldots, B_m)$

$R_2(A_1, \ldots, A_n, C_1, \ldots, C_p)$

If $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$ or $A_1, \ldots, A_n \rightarrow C_1, \ldots, C_p$
Then the decomposition is lossless

Note: don’t need both

- Examples:
- name → price, so first decomposition was *lossless*
- category $\not\rightarrow$ name and category $\not\rightarrow$ price, and so second decomposition was *lossy*

Quick lossless/lossy example

X	Y	Z
1	2	3
4	2	5

- At a glance: can we decompose into $R_1(Y,X)$, $R_2(Y,Z)$?
- At a glance: can we decompose into $R_1(X,Y)$, $R_2(X,Z)$?

Next topic: Normal Forms

- First Normal Form = all attributes are atomic
  - As opposed to set-valued
  - Assumed all along
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- **Boyce Codd Normal Form (BCNF)**
- Fourth Normal Form (4NF)
- Fifth Normal Form (5NF)

BCNF definition

- A simple condition for removing anomalies from relations:

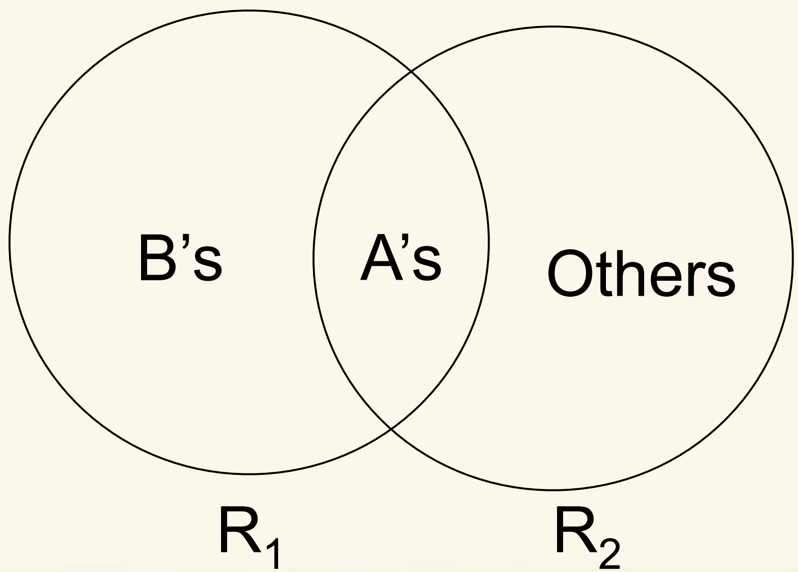
> A relation R is in BCNF if:
>
> If **As → Bs** is a **non-trivial** dependency
>
> in R , then As is a superkey for R

- I.e.: The left side must always contain a key
- I.e: If a set of attributes determines other attributes, it must determine *all* the attributes

- Slogan: “In every FD, the left side is a superkey.”

BCNF decomposition algorithm

Repeat
  choose $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ that violates the BNCF condition
  //Heuristic: choose Bs as large as possible
  split R into $R_1(A_1, \ldots, A_m, B_1, \ldots, B_n)$ and $R_2(A_1, \ldots, A_m, [others])$
  continue with both R_1 and R_2
Until no more violations



Boyce-Codd Normal Form

- Name/phone example is not BCNF:

Name	SSN	Mailing-address	Phone
Michael	123	NY	212-111-1111
Michael	123	NY	917-111-1111

  - {**ssn,phone**} is key
  - FD: **ssn → name,mailing-address** holds
    - Violates BCNF: **ssn** is not a superkey
- Its decomposition is BCNF
  - Only superkeys → anything else

Name	SSN	Mailing-address
Michael	123	NY

SSN	PhoneNumber
123	212-111-1111
123	917-111-1111

Design/BCNF example

- Consider situation:
  - Entities: Emp(ssn,name,lot), Dept(id,dname,budg)
  - Relship: Works(E,D,since)
- *Draw E/R*

- New rule: in each dept, everyone parks in same lot
- *Translate to FD*
- *Normalize*

BCNF Decomposition

- Larger example: multiple decompositions
- {Title, Year, Studio, President, Pres-Address}
- FDs:
  - Title, Year → Studio
  - Studio → President
  - President → Pres-Address
  - => Studio → President, Pres-Address
- No many-many this time
- Problem cause: *transitive FDs*:
  - Title, year → studio → president

BCNF Decomposition

- Illegal: As → Bs, where As not a superkey
- Decompose: **Studio → President, Pres-Address**
  - As = {studio}
  - Bs = {president, pres-address}
  - Cs = {title, year}
- Result:
  1. Studios(studio, president, pres-address)
  2. Movies(studio, title, year)
- Is (2) in BCNF? Is (1) in BCNF?
  - Key: Studio
  - FD: President → Pres-Address
  - Q: Does president → studio? If so, president is a key
  - *But if not*, it violates BCNF

BCNF Decomposition

- Studios(<u>studio</u>, president, pres-address)
- Illegal: **As → Bs**, where As not a superkey
- ➔ Decompose: President → Pres-Address
  - As = {president}
  - Bs = {pres-address}
  - Cs = {studio}
- {<u>Studio</u>, President, Pres-Address} becomes
  - {<u>President</u>, Pres-Address}
  - {<u>Studio</u>, President}

Decomposition algorithm example

- R(N,O,R,P)   F = {N → O, O → R, R → N}

Name	Office	Residence	Phone
George	Pres.	WH	202-…
George	Pres.	WH	486-…
Dick	VP	NO	202-…
Dick	VP	NO	307-…

- Key: N,P
- Violations of BCNF: N → O, O→R, N →OR

- Pick N → OR (on board)
- Can we rejoin? (on board)
- What happens if we pick N → O instead?
- Can we rejoin? (on board)

An issue with BCNF

- We could lose FDs

- Relation: R(Title, Theater, Neighborhood)
- FDs:
  - Title,N’hood → Theater
    - *Assume a movie shouldn’t play twice in same n’hood*
  - Theater → N’hood
- Keys:
  - {Title, N’hood}
  - {Theater, Title}

Title	Theater	N’hood
阿凡达	飞扬	天河
碟中碟	飞扬	天河

Losing FDs

- BCNF violation: Theater → N'hood
- Decompose:
  - {Theater, N'Hood}
  - {Theater, Title}

- Resulting relations:

R1

Theater	N'hood
飞扬	天河

R2

Theater	Title
飞扬	阿凡达
飞扬	碟中碟

Losing FDs

- Suppose we add new rows to R1 and R2:

R1

Theater	N'hood
飞扬	天河
天娱	天河

R2

Theater	Title
飞扬	碟中碟
飞扬	阿凡达
天娱	碟中碟

R'

Theater	N'hood	Title
飞扬	天河	碟中碟
飞扬	天河	阿凡达
天娱	天河	碟中碟

- Neither R1 nor R2 enforces FD *Title,N'hood* → *Theater*

Third normal form: motivation

- Sometimes
  - BCNF is *not dependency-preserving*, and
  - Efficient checking for FD violation on updates is important
- In these cases BCNF is too severe a req.
  - “over-normalization”
- Solution: define a weaker normal form, 3NF
  - FDs can be checked on individual relations without performing a join (no inter-relational FDs)
  - relations can be converted, preserving both data and FDs

BCNF lossiness

- 注意: BCNF decomp. is *not data-lossy*
  - Results can be rejoined to obtain the exact original

- But: it *can* lose *dependencies*
  - *After decomp*, now legal to add rows whose *corresponding rows* would be illegal in (rejoined) original

- Data-lossy v. FD-lossy

Third Normal Form

- Now define the (weaker) Third Normal Form
  - Turns out: this example was already in 3NF

A relation R is in 3rd normal form if :

For every nontrivial dependency $A_1, A_2, ..., A_n \rightarrow B$ for R, $\{A_1, A_2, ..., A_n\}$ is a super-key for R, or B is part of a key, i.e., B is *prime*

Tradeoff:

BCNF = no FD anomalies, but may lose some FDs
3NF = keeps all FDs, but may have some anomalies

Canonical Cover

- Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F.
  - Attribute A is extraneous in α if $A \in \alpha$ and if A is removed from α, the s
    functional dependencies implied by F doesn't change.
    Given A~~B~~ $\rightarrow$ C *and* A $\rightarrow$ C then B is extraneous in AB
  - Attribute A is extraneous in β if $A \in \beta$ and if A is removed from β, the set of functional dependencies implied by F doesn't change.
    Given A $\rightarrow$ ~~B~~C *and* A $\rightarrow$ B then B is extraneous in BC

From A $\rightarrow$ C
I get AB $\rightarrow$ C

- A canonical cover F_c for F is a set of dependencies such that F logically implies all dependencies in F_c and F_c logically implies all dependencies in F, and further
  - No functional dependency in F_c contains an extraneous attribute.
  - Each left side of a functional dependency in F_c is unique.

- 注意：**Canonical Cover** 与书本的最小函数依赖集不完全一样

Canonical Cover

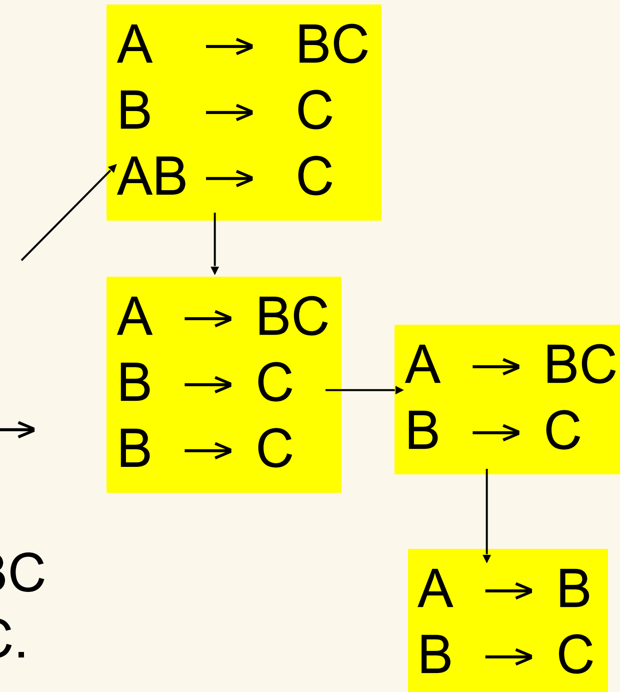
- Compute a canonical over for F:
  repeat
  use the union rule to replace any dependencies in F
  $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ replaced with $\alpha_1 \rightarrow \beta_1\beta_2$

- Find a functional dependency $\alpha \rightarrow \beta$ with an
  extraneous attribute either in α or in β
  If an extraneous attribute is found, delete it from
  $\alpha \rightarrow \beta$
  until F does not change

Example of Computing a Canonical Cover

- $R = (A, B, C)$
  $F = \{ \quad A \rightarrow BC$
  $B \rightarrow C$
  $A \rightarrow B$
  $AB \rightarrow C\}$
- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
- A is extraneous in $AB \rightarrow C$ because $B \rightarrow C$ logically implies $AB \rightarrow C$.
- C is extraneous in $A \rightarrow BC$ since $A \rightarrow BC$ is logically implied by $A \rightarrow B$ and $B \rightarrow C$.
- The canonical cover is:

  $A \rightarrow B$
  $B \rightarrow C$



3NF Decomposition Algorithm

- Let F_c be a canonical cover for F;
  i := 0;
  for each functional dependency $\alpha \rightarrow \beta$ in F_c do
      if none of the schemas R_j, $1 <= j <= i$ contains $\alpha\beta$
          then begin
              i:=i+1;
              R_j:= $\alpha\beta$;
            end
  if none of the schemas R_j, $1 <= j <= i$ contains a candidate key for R
      then begin
          i:=i+1;
          R_i:= any candidate key for R;
        end
      return (R_1, R_2, …, R_i)

Example

- Relation schema:
  Banker-info-schema
  branch-name, customer-name, banker-name, office-number
- The functional dependencies for this relation schema are:
  banker-name → branch-name, office-number
  customer-name, branch-name → banker-name
- The key is:
  {customer-name, branch-name}

Applying 3NF to banker - info - schema

- Go through the for loop in the algorithm:

  banker-name → branch-name, office-number
  is not in any decomposed relation (no decomposed relation so far)
  Create a new relation:
  Banker-office-schema (banker-name, branch-name, office-number)

  customer-name, branch-name → banker-name
  is not in any decomposed relation (one decomposed relation so far) Create a new relation:
  Banker-schema (customer-name, branch-name, banker-name)
- Since Banker-schema contains a candidate key for Banker-info-schema, we are done with the decomposition process.

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into relations in 3NF and
  - the decomposition is lossless
  - dependencies are preserved
- It is always possible to decompose a relation into relations in BCNF and
  - the decomposition is lossless
  - it may not be possible to preserve dependencies

Next week

- Check course homepage for homework.
- Read ch.4.1-2 (RA)