Normalization 3NF and BCNF

CS 4750 Database Systems

[A. Silberschatz, H. F. Korth, S. Sudarshan, Database System Concepts, Ch.7]
[Ricardo and Urban, Database Illuminated, Ch.6]
[https://www.w3schools.in/dbms/database-normalization/]

Recap: Normalization

Goals:

- Reduce redundancy
- Improve data integrity
- Queries are logical and efficient

Decomposition: Three properties that must satisfied

- **Lossless join decomposition** – avoid data corruption
  - No gain/no loss
- **Dependency preserving** – improve performance
  - No joins needed to check a dependency
- **Remove duplication** – keep size and structure of DB stable
  - Minimize redundant data in a table

3NF and Decomposition

- **Lossless-join**
- Always **dependency preserving**
- Possible to have extra data (there may be redundancy)

Questions:
Is the relation in 3NF?
Is any refinement needed?

To calculate 3NF

- Identify PK of the original table
- Take Canonical Cover (Fc)
- Turn (minimal set of) FDs into tables

Canonical Cover (Fc)

- A minimal set of functional dependencies that has the same closure as the original set F
- Extraneous attributes = attribute of FDs that we can removed without changing the closure of FDs

- F logically implies all dependencies in Fc
- Fc logically implies all dependencies in F
- No FD in Fc contains an extraneous attribute

F and F+ are logically equivalent

Minimal basis for a set of FDs: For any set of FDs, there is at least one minimal basis, which is a set of FDs equivalent to the original (each set implies the other set), with singleton right sides, no FD that can be eliminated while preserving equivalence, and no attribute in a left side that can be eliminated while preserving equivalence

Canonical Cover (Fc)

Compute the canonical cover of a set of functional dependencies F

Always start with F and use rules to minimize

$Fc = F$
repeat
  apply union rule to replace any dependencies $f: X_1 \rightarrow Y_1$
    and $f: X_1 \rightarrow Y_2$ with $f: X_1 \rightarrow Y_1Y_2$
  for each functional dependency f_i
    if f_i contains an extraneous attribute either in X or in Y
      then remove an extraneous attribute
until Fc does not change any further

Example 1: 3NF and Fc

Let's do this together

Given R(A,B,C,D,E)

FDs = { A → B, AB → D, B → BDE, C → D, D → D }

Compute Fc and convert the relation into 3NF

Observation: AC is a minimal super key of the given R

(1) write all LHS

A →
B →
AB →
C →
D →

(2) copy FDs as is

A → B
B → B DE
AB → D
C → D
D → D

(3) remove reflexivity

A → B
B → ~~B~~ DE
AB → D
C → D
~~D → D~~

(4) remove extraneous attr

A → B
B → DE
~~AB → D~~
C → D

A → B and B → D.
Thus, remove AB → D

Example 1: 3NF and Fc

Let's do this together

Super key

(from previous page)

(4) Remove extraneous attr

A	→	B
B	→	DE
C	→	D

Turn Fc into tables

AC (a minimal super key of the original R)

AB

BDE

CD

A relation R(A, B, C, D, E) is converted into 3NF by putting LHS and RHS of each FD in Fc together in one relation.

Dependency preserving

R(A, B, C, D, E) becomes R_1(A, C), R_2(A, B), R_3(B, D, E), R_4(C, D)

Or write it in another format: AC // AB // BDE // CD

Example 2: 3NF and Fc

Let's do this together

Given R(A,B,C)

FDs = { A → BC, B → C, A → B, AB → C }

Compute Fc and convert the relation into 3NF

---

(1) write all LHS

A →
B →
AB →

(2) copy FDs as is

A → BC
B → C
AB → C

Combine the given FDs A → BC and A → B

No reflexivity to remove

(3) remove extraneous attr

A → BC
B → C
~~AB → C~~

Consider AB → C and B → C,
A is an extraneous attr, remove A from AB → C (resulting in B → C)

(4) remove extraneous attr

A → B~~C~~
B → C

Apply decomposition to A → BC, thus, A → B and A → C.

A → C is logically equivalent to A → B and B → C (transitivity).

Thus, C is an extraneous attr, remove C from A → BC

Example 2: 3NF and Fc

Let’s do this together

Super key

(from previous page)

(4) Remove extraneous attr

A	→	B
B	→	C

Turn Fc into tables

AB
BC

R(A, B, C) becomes R_1(A, B) and R_2(B, C)

Or write it in another format: AB // BC

BCNF and Decomposition

- **Lossless-join**
- Guarantee **redundancy free**
- May involve dependency across relations

Given a relation R,

for every nontrivial FD X → Y in R, X is a super key

- For all FDs, "key → everything"

Questions:
Is the relation in BCNF?
Is any refinement needed?

BCNF and Decomposition

To calculate BCNF

Compute *F+*
repeat given a relation *R* (or a decomposed *R*) and FDs *F*
  for each functional dependency f_i in a relation *R*
    if f_i violates $X \rightarrow Y$
      then decompose *R* into two relations:
        one with X U Y as its attributes (i.e., everything *f*)
        one with X U (attrs(*R*) – *X* – *Y*) as its attributes
until no violation

Example: BCNF and F+

Let's do this together

Given R(A,B,C,D,E)

FDs = { A → B, AB → D, B → BDE, C → D, D → D }

Compute F+ and convert the relation into BCNF

Compute F+

(1) write all LHS & remaining	(2) copy FDs as is	(3) apply reflexivity	(4) apply transitivity
A →	A → B	A → AB	A → AB DE
B →	B → B DE	B → B DE	B → B DE
AB →	AB → D	AB → AB D	AB → AB DE
C →	C → D	C → CD	C → CD
D →	D → D	D → D	D → D
E →	E →	E → E	E → E

Example: BCNF and F+

Let's do this together

(from previous page)

(4) apply transitivity

F+

A	→	AB DE
B	→	B DE
AB	→	AB DE
C	→	CD
D	→	D
E	→	E

Not trivial FDs. A, B, C are not super key

Trivial FDs (let's abbreviate to TR)

Based on F+, let's rewrite using the following format to help us calculate

R (A B C D E)

! ! ! TR TR

TR	– trivial
SK	– super key
X	– neither trivial nor super key
!	– (possibly) need to work on

Example: BCNF and F+

R ($\boxed{A}$ B C ~~D~~ ~~E~~)

! ! ! TR TR

Let's do this together

F+

A	→	AB DE
B	→	B DE
AB	→	AB DE
C	→	CD
D	→	D
E	→	E

To choose which FD to work on, two ways:

- Choose the first FD, or
- Choose the longest FD (yield better solution)

Let's consider A:

A is not a super key, not trivial, thus A → ABDE violates BCNF, break a relation on A

Example: BCNF and F+

Let's do this together

R (A $\boxed{\text{B}}$ C ~~D~~ ~~E~~)

! ! ! TR TR

F+

A	→	AB DE
B	→	B DE
AB	→	AB DE
C	→	CD
D	→	D
E	→	E

A → ABDE

Take *RHS*, make a table: ABDE
Take *LHS*, make a table where A is a key
A plus (original – (*RHS*)) --- thus, AC

There are only 2 attrs, this relation is ok

Break on A ABCDE

ABDE

AC ok

Verify table ABDE if there is any violation.

A is a super key. Also can't break on A twice. Still need to work on B.

Restriction: Cannot break on A 2 times in a row

Next: consider B. B is neither trivial nor super key, break on B

Example: Calculate BCNF

Let's do this together

R (A B ~~D~~ ~~E~~)

X ! TR TR

F+

A	→	AB DE
B	→	B DE
AB	→	AB DE
C	→	CD
D	→	D
E	→	E

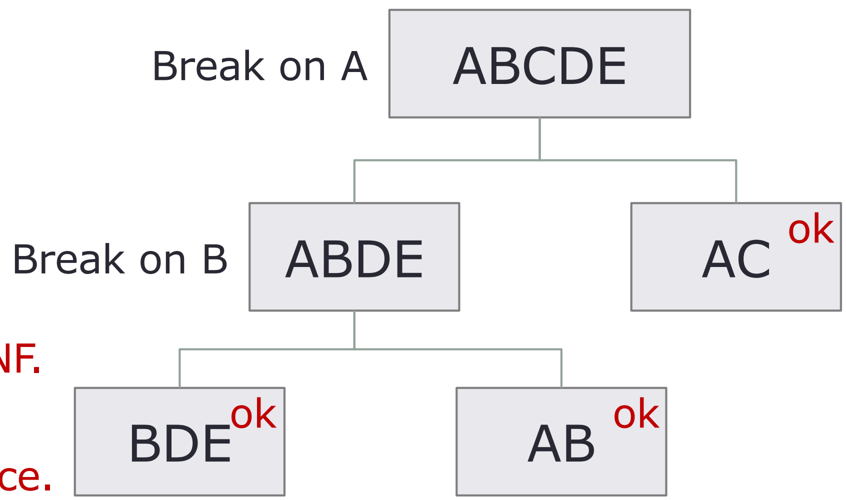
B → BDE

Take *RHS*, make a table: BDE

Take *LHS*, make a table where B is a key

B plus (original – (*RHS*)) --- thus, AB

There are only 2 attrs, this relation is ok



Verify that B → BDE does not violate BCNF.

B is super key. Also, can't break on B twice.

D and E are trivial.

R(ABCDE) becomes

AC // AB // BDE

(notice: results are different 3NF)

Wrap-Up

- Compute F+ and Fc
- 3NF and decomposition
- BCNF and decomposition

What's next?

- SQL

Additional practice

Practice 1: Decomposition

Given R (A, B, C)
FDs = { A → B, B → C }

Supposed R is decomposed in two different ways :

1. R1(A, B), R2(B, C)
   - Does this satisfy lossless-join decomposition?

     Yes: R1 ∩ R2 = {B} and B → BC (B is super key of R2)
   - Does this satisfy dependency preserving?

     Yes: dependencies can be checked without joining tables

2. R1(A, B), R2(A, C)
   - Does this satisfy lossless-join decomposition?

     Yes: R1 ∩ R2 = {A} and A → AB (A is super key of R1)
   - Does this satisfy dependency preserving?

     No: cannot check B → C without joining R1 and R2

Practice 2: 3NF and BCNF

Given R (A, B, C, D, E)
FDs = { A → C, C → DE, D → B, A → D }

Decompose table R using 3NF

Fc:
- A → C
- C → DE
- D → B

AC // BD // CDE

Decompose table R using BCNF

F+:
- A → ABCDE
- B → B
- C → BCDE
- D → BD
- E → E

AC // BD // CDE

Practice 3: 3NF

Does the Customer_order table satisfy 3NF requirements?
If not, convert the table into 3NF

Customer_order

OrderId	CustomerID	Date	Store	Address
1	2	10/1/2019	South	11 Sorth Str
2	1	9/25/2019	West	22 West Str
3	3	8/12/2019	East	33 East Str
4	4	10/23/2019	West	22 West Str
5	8	5/11/2019	North	44 North Str
6	6	5/11/2019	South	11 Sorth Str
7	5	7/31/2019	East	33 East Str
8	7	10/17/2019	West	22 West Str
9	6	9/19/2019	North	44 North Str
10	4	10/23/2019	North	44 North Str
…	…	…	…	…

Practice 3: 3NF - solution

No. OrderId → Store → Address

"Transitive dependency"

Customer_order

OrderId	CustomerID	Date	Store
1	2	10/1/2019	South
2	1	9/25/2019	West
3	3	8/12/2019	East
4	4	10/23/2019	West
5	8	5/11/2019	North
6	6	5/11/2019	South
7	5	7/31/2019	East
8	7	10/17/2019	West
9	6	9/19/2019	North
10	4	10/23/2019	North
…	…	…	…

Store

Store	Address
South	11 Sorth Str
West	22 West Str
East	33 East Str
North	44 North Str

Practice 4: BCNF

Does the Student_Major_Advisor table satisfy BCNF requirements?
If not, convert the table into BCNF

Student_Major_Advisor

ComputingID	Major	Advisor
ht1y	Computer Science	someone1
dt2y	Physics	someone2
dt2y	Engineering	somoone3
md3y	Computer Science	someone4
mn4e	Math	someone5
md3y	Computer Science	someone1

computingID, Major → Advisor

Advisor → Major

Assume:
(semantic/business rules)

- Each Student may major in several subjects.
- For each Major, a given Student has only one Advisor.
- Each Major has several Advisors.
- Each Advisor advises only one Major.
- Each Advisor advises several Students in one Major.

Practice 4: BCNF - solution

No. Contain non-key attribute(s)

Student_Advisor

ComputingID	Advisor
ht1y	someone1
dt2y	someone2
dt2y	somoone3
md3y	someone4
mn4e	someone5
md3y	someone1

Advisor_Major

Advisor	Major
someone1	Computer Science
someone2	Physics
somoone3	Engineering
someone4	Computer Science
someone5	Math