Normal Forms

Functional dependencies

Our goal:
given a set of FD set, F, find an alternative FD set, G that is:
smaller
equivalent

Bad news:
Testing F=G (F+ = G+) is computationally expensive

Good news:
Minimal Cover (or Canonical Cover) algorithm:
given a set of FD, F, finds minimal FD set equivalent to F

Minimal: can't find another equivalent FD set w/ fewer FD's

Minimal Cover Algorithm

Given:

F = { A → BC,
B → CE,
A → E,
AC→ H,
D → B}

Determines minimal cover of F:

Fc = { A → BH,
B → CE,
D → B}

- Fc = F
- No G that is equivalent to F and is smaller than Fc

Another example:

F = { A → BC,
B → C,
A → B,
AB→ C,
AC → D}

⟶ MC Algorithm ⟶

Fc = { A → BD,
B →C}

Minimal Cover Algorithm

Basic Algorithm

ALGORITHM MinimalCover (X: FD set)
BEGIN
REPEAT UNTIL STABLE
(1) Where possible, apply UNION rule (A's axioms)
(e.g., A →BC, A→CD becomes A→BCD)
(2) remove "extraneous attributes" from each FD
(e.g., AB→C, A→B becomes
A→B, B→C
i.e., A is extraneous in AB→C)

Extraneous Attributes

(1) Extraneous is RHS?
e.g.: can we replace A → BC with A→C?
(i.e. Is B extraneous in A →BC?)

(2) Extraneous in LHS ?
e.g.: can we replace AB →C with A → C ?
(i.e. Is B extraneous in AB→C?)

Simple but expensive test:
1. Replace A → BC (or AB → C) with A → C in F

F2 = F - {A →BC} U {A →C}
or
F - {AB→C} U {A →C}

2. Test if F2+ = F+ ?
if yes, then B extraneous

Extraneous Attributes

A. RHS: Is B extraneous in A →BC?

step 1: F2 = F - {A →BC} U {A →C}
step 2: F+ = F2+ ?

To simplify step 2, observe that F2+ $\subseteq$ F+

i.e., not new FD's in F2+)

Why? Have effectively removed A→B from F

When is F+ = F2+ ?
Ans. When (A→B) in F2+

Idea: if F2+ includes: A→B and A→C,
then it includes A→BC

Extraneous Attributes

B. LHS: Is B extraneous in A B→C ?

step 1: F2 = F - {AB →C} U {A →C}
step 2: F+ = F2+ ?

To simplify step 2, observe that F+ $\subseteq$ F2+

i.e., there may be new FD's in F2+)

Why? A→C "implies" AB→C. therefore all FD's in F+ also in F2+.
But AB→C does not "imply" A→C

When is F+ = F2+ ?

Ans. When (A→C) in F+

Idea: if F+ includes: A→C then it will include all the FD's of F+.

Extraneous attributes

A. RHS :
Given F = {A→BC, B→C} is C extraneous in A →BC?

why or why not?

Ans: yes, because

A→C in { A→B, B→C}+

Proof.
1. A→ B
2. B →C
3. A→C transitivity using Armstrong's axioms

Extraneous attributes

B. LHS :

Given F = {A→B, AB→C} is B extraneous in AB →C?

why or why not?

Ans: yes, because

A→C in F+

Proof. 1. A→ B
2. AB →C
3. A→C using pseudotransitivity on 1 and 2

Actually, we have AA→C but {A, A} = {A}

Minimal Cover Algorithm

ALGORITHM MinimalCover (F: set of FD's)
BEGIN
REPEAT UNTIL STABLE
(1) Where possible, apply UNION rule (A's axioms)

(2) Remove all extraneous attributes:
a. Test if B extraneous in A→ BC
(B extraneous if
(A→B) in (F - {A→BC} U {A→C})+)
b. Test if B extraneous in AB→C
(B extraneous in AB→C if
(A→C) in F+)

Minimal Cover Algorithm

Example: determine the minimal cover of

F = {A →BC, B→CE, A→E}

Iteration 1:

a. F = { A→BCE, B→CE}
b. Must check for up to 5 extraneous attributes

- B extraneous in A→BCE? No
- C extraneous in A → BCE?
  yes: (A→C) in { A→BE, B→CE}
  1. A→BE -> 2. A→B -> 3. A→CE -> 4. A → C

- E extraneous in A→BE?

Minimal Cover Algorithm

Example: determine the minimal cover of

F = {A →BC, B→CE, A→E}

Iteration 1:

a. F = { A→BCE, B→CE}
b. Must check for up to 5 extraneous attributes

- B extraneous in A→BCE? No
- C extraneous in A → BCE? Yes
- E extraneous in A→BE?
  1. A→B -> 2. A→CE -> A→E
- E extraneous in B→CE No
- C extraneous in B→CE No

Iteration 2:

a. F = { A → B, B→ CE}
b. Extraneous attributes:
  - C extraneous in B → CE No
  - E extraneous in B →CE No

DONE

Minimal Cover Algorithm

Find the minimal cover of

F = { A → BC,
B → CE,
A → E,
AC → H,
D → B}

Ans: Fc= { A→BH, B→ CE, D→B}

Minimal Cover Algorithm

Find two different minimal covers of:

F = { A→BC, B→ CA, C→AB}

Ans:

Fc1 = { A →B, B→C, C→A}

and

Fc2 = { A→C, B→A, C→B}

FD so far...

1. Minimal Cover algorithm
   • result (Fc) guaranteed to be the minimal FD set equivalent to F

2. Closure Algorithms
   a. Armstrong's Axioms:
      more common use: test for extraneous attributes
      in C.C. algorithm
   b. Attribute closure:
      more common use: test for superkeys

3. Purposes
   a. minimize the cost of global integrity constraints
      so far: min gic's = |Fc|

      In fact.... Min gic's = 0
      (FD's for "normalization")

Another use of FD's: Schema Design

Example:

R =

bname	bcity	assets	cname	lno	amt
Downtown	Bkln	9M	Jones	L-17	1000
Downtown	Bkln	9M	Johnson	L-23	2000
Mianus	Horse	1.7M	Jones	L-93	500
Downtown	Bkln	9M	Hayes	L-17	1000

R: "Universal relation"
tuple meaning: Jones has a loan (L-17) for $1000 taken out at the Downtown branch in Bkln which has assets of $9M

Design:
+ : fast queries (no need for joins!)
- : redudancy:
  update anomalies   examples?
  deletion anomalies

Decomposition

1. Decomposing the schema
   R = (bname, bcity, assets, cname, lno, amt)

R = R1 U R2

R1 = (bname, bcity, assets, cname)   R1 = (cname, lno, amt)

2. Decomposing the instance

bname	bcity	assets	cname	lno	amt
Downtown	Bkln	9M	Jones	L-17	1000
Downtown	Bkln	9M	Johnson	L-23	2000
Mianus	Horse	1.7M	Jones	L-93	500
Downtown	Bkln	9M	Hayes	L-17	1000

bname	bcity	assets	cname
Downtown	Bkln	9M	Jones
Downtown	Bkln	9M	Johnson
Mianus	Horse	1.7M	Jones
Downtown	Bkln	9M	Hayes

cname	lno	amt
Jones	L-17	1000
Johnson	L-23	2000
Jones	L-93	500
Hayes	L-17	1000

Goals of Decomposition

1. Lossless Joins
   Want to be able to reconstruct big (e.g. universal) relation by joining smaller ones (using natural joins)
   (i.e. $R1 \bowtie R2 = R$)

2. Dependency preservation
   Want to minimize the cost of global integrity constraints based on FD's
   (i.e. avoid big joins in assertions)

3. Redundancy Avoidance
   Avoid unnecessary data duplication (the motivation for decomposition)

Why important?

LJ : information loss
DP: efficiency (time)
RA: efficiency (space), update anomalies

Dependency Goal #1: lossless joins

A bad decomposition:

bname	bcity	assets	cname
Downtown	Bkln	9M	Jones
Downtown	Bkln	9M	Johnson
Mianus	Horse	1.7M	Jones
Downtown	Bkln	9M	Hayes

$\bowtie$

cname	lno	amt
Jones	L-17	1000
Johnson	L-23	2000
Jones	L-93	500
Hayes	L-17	1000

=

	bname	bcity	assets	cname	lno	amt
	Downtown	Bkln	9M	Jones	L-17	1000
→	Downtown	Bkln	9M	Jones	L-93	500
	Downtown	Bkln	9M	Johnson	L-23	2000
→	Mianus	Horse	1.7M	Jones	L-17	1000
	Mianus	Horse	1.7M	Jones	L-93	500
	Downtown	Bkln	9M	Hayes	L-17	1000

Problem: join adds meaningless tuples
"lossy join": by adding noise, have lost meaningful information as a result of the decomposition

Dependency Goal #1: lossless joins

Is the following decomposition lossless or lossy?

bname	assets	cname	lno
Downtown	9M	Jones	L-17
Downtown	9M	Johnson	L-23
Mianus	1.7M	Jones	L-93
Downtown	9M	Hayes	L-17

lno	bcity	amt
L-17	Bkln	1000
L-23	Bkln	2000
L-93	Horse	500

Ans: Lossless: R = R1 $\bowtie$ R2, it has 4 tuples

Ensuring Lossless Joins

A decomposition of R : R = R1 U R2 is lossless iff

$R1 \cap R2 \rightarrow R1$, or

$R1 \cap R2 \rightarrow R2$

(i.e., intersecting attributes must be a superkey for one of the resulting smaller relations)

Decomposition Goal #2: Dependency preservation

Goal: efficient integrity checks of FD's

An example w/ no DP:
R = (bname, bcity, assets, cname, lno, amt)
bname → bcity assets
lno → amt bname

Decomposition: R = R1 U R2
R1 = (bname, assets, cname, lno)
R2 = (lno, bcity, amt)

Lossless but not DP. Why?

Ans: bname →bcity assets crosses 2 tables

Decomposition Goal #2: Dependency preservation

To ensure best possible efficiency of FD checks

ensure that only a SINGLE table is needed in order to check each FD

i.e. ensure that: A1 A2 ... An → B1 B2 ... Bm

Can be checked by examining Ri = (..., A1, A2, ..., An, ..., B1, ..., Bm, ...)

To test if the decomposition R = R1 U R2 U ... U Rn is DP

(1) see which FD's of R are covered by R1, R2, ..., Rn

(2) compare the closure of (1) with the closure of FD's of R

Decomposition Goal #2: Dependency preservation

Example: Given F = { A→B, AB→ D, C→ D}

consider R = R1 U R2 s.t.
R1 = (A, B, C) , R2 = (C, D) is it DP?

(1) F+ = { A→BD, C→D}+
(2) G+ = {A→B, C→D} +

(3) F+ = G+ ? No because (A→D) not in G+

Decomposition is not DP

Decomposition Goal #2: Dependency preservation

Example: Given F = { A→B, AB→ D, C→ D}

consider R = R1 U R2 s.t.
R1 = (A, B, D) , R2 = (C, D)

(1) F+ = { A→BD, C→D}+
(2) G+ = {A→BD, C→D, ...} +

(3) F+ = G+
note: G+ cannot introduce new FDs not in F+

Decomposition is DP

Decomposition Goal #3: Redudancy Avoidance

Example:

A	B	C
a	x	1
e	x	1
g	y	2
h	y	2
m	y	2
n	z	1
p	z	1

Redundancy for B=x , y and z

(1) An FD that exists in the above relation is: B → C

(2) A superkey in the above relation is A, (or any set containing A)

When do you have redundancy?
Ans: when there is some FD, X→Y covered by a relation and X is not a superkey

Normalization

Decomposition techniques for ensuring:

Lossless joins

Dependency preservation

Redundancy avoidance

We will look at some normal forms:

Boyce-Codd Normal Form (BCNF)

3rd Normal Form (3NF)

Boyce-Codd Normal Form (BCNF)

What is a normal form?

Characterization of schema decomposition
in terms of properties it satisfies

BCNF: guarantees no redundancy

Defined:
relation schema R, with FD set, F is in BCNF if:

For all nontrivial X→Y in F+:
X→R (i.e. X a superkey)

BCNF

Example: R=(A, B, C)
F = (A→B, B→C)

Is R in BCNF?

Ans: Consider the non-trivial dependencies in F+:

A→B, A→R (A a key)
A→C, -//-
B→C, B-/-> R (B not a superkey)

Therefore not in BCNF

BCNF

Example:
R= R1 U R2
R1 = (A, B) , R2 = (B,C)
F = (A→B, B→C)

Are R1, R2 in BCNF?

Ans: Yes, both non-trivial FDs define a key in R1, R2

Is the decomposition lossless? DP?

Ans: Losless: Yes. DP: Yes.

BCNF

Decomposition Algorithm

Algorithm BCNF(R: relation, F: FD set)

Begin
1. Compute F+
2. Result → {R}
3. While some R_i in Result not in BCNF Do
   a. Chose (X→Y) in F+ s.t.
      (X→Y) covered by R_i
      X -/-> R_i (X not a superkey for R_i)
   b. Decompose R_i on (X→Y)
      R_{i1} ← X U Y
      R_{i2} ← R_i - Y
   c. Result ← Result - {R_i} U { R_{i1}, R_{i2}}

4. return Result

End

BCNF Decomposition

Example:

R= (A, B, C, D)
F = (A→B, AB→D, B→C)

Decompose R into BCNF

Ans: Fc = {A →BD, B→C}
R=(A, B, C, D)
B→ C is covered by R and B not a superkey

R1 = (B,C)
In BCNF:
B→C and B key

R2 = (A, B, D)
In BCNF: A→B, A→D, A→BD
and A is a key

BCNF Decomposition

Example:

R = (bname, bcity, assets, cname, lno, amt)
F = { bname →bcity assets,
lno → amt bname}

key= superkey here

Decompose R into BCNF   Ans: Fc = F

R , and *bname → bcity* covered by R, bname not a key

R1 = (bname, bcity)
In BCNF

R2 = (bname, assets, cname, lno, amt)
lno → amt bname covered by R2 and lno not a key

R3 = (lno, amt, bname)
In BCNF

R4 = (assets, cname, lno)
lno → assets ....

R5 = (lno, assets)

R6=(lno, cname)

Not DP! bname →assets is not covered by any relation AND cannot be implied by the covered FDs. Covered FDs: G = { bname → bcity, lno → amt bname, lno → assets}

BCNF Decomposition

Can there be > 1 BCNF decompositions?

Ans: Yes, last example was not DP. But...

Given Fc = { bname → bcity assets, lno → amt bname}

R=(bname, bcity, assets, cname, lno, amt)
bname → bcity assets and bname -/-> R

R1 = (bname, bcity, assets)
BCNF: bname →R1

R2 = (bname, cname, lno, amt)
lno → amt bname, lno -/-> R2

R3 = (lno, amt, bname)
BCNF: lno → R3

R4 = (lno, cname)

Is R = R1 U R3 U R4 DP? → Yes!!

BCNF Decomposition

Can we decompose on FD's in Fc to get a DP, BCNF decomposition?

Usually, yes, but ...

Consider: R = (J, K, L)
F = (JK→L, L→K} (Fc = F)

We can apply decomposition either using: JK→L , L→K or the oposite

Dec. #1	Dec. #2
Using L→K	Using JK→L
R1 = (L, K) R2 = (J, L) Not DP.	R1 = (J, K, L) not BCNF R2 = (J, K)

So, BCNF and DP decomposition may not be possible.

Aside

Is the example realistic?

Consider: BankerName → BranchName
BranchName CustomerName → BankerName

3NF: An alternative to BCNF

Motivation:
sometimes, BCNF is not what you want

E.g.: street city → zip and zip → city
BCNF: R1 = { zip, city} R2 ={ zip, street}

No redundancy, but to preserve 1st FD requires assertion with join

Alternative: 3rd Normal Form
Designed to say that decomposition can stop at {street, city, zip}

3NF: An alternative to BCNF

BCNF test: Given R with FD set, F: For any non-trivial FD,
X→Y in F+ and covered by R, then X→R

3NF test: Given R with FD set, F:
For any non-trivial FD,
X→Y in F+ and covered by R, then
X→ R or
Y is a subset of some candidate key of R

Thus, 3NF a weaker normal form than BCNF:

i.e. R in BCNF => R in 3NF
but R in 3NF =/=> R in BCNF (not sure than R is in BCNF)

3NF: An alternative to BCNF

Example:

R=(J, K, L)   F = {JK→L, L→K}

then R is 3NF!

Key for R: JK

JK→L covered by R, JK→R
L→K, K is a part of a candidate key

3NF

Example:

R=(bname, cname, lno, amt)
F=Fc = { lno→ amt bname,
cname bname → lno}

Q: is R in BCNF, 3NF or neither?

Ans:

R not in BCNF: lno → amt , covered by R and lno -/->R

R not in 3NF: candidate keys of R: lno cname
or
cname bname

lno → amt bname covered by R
{amt bname} not a subset of a candidate key

3NF

Example: R = R1 U R2
R1 = (lno, amt, bname)
R2 = (lno, cname, bname), F=Fc = { lno→ amt bname, cname bname → lno}

Q: Are R1, R2 in BCNF, 3NF or neither?

Ans: R1 in BCNF : lno→amt bname covered by R1 and lno →R1

R2 not in BCNF: lno →bname and lno-/-> R2

R1 in 3NF (since it is in BCNF)

R2 in 3NF: R2's candidate keys: cname bname and lno cname

lno → bname, bname subset of a c.key
cname bname → lno , lno subset of a c. key

3NF Decomposition Algorithm

Algorithm 3NF (R: relation, F: FD set)

1. Compute Fc
2. i ← 0
3. For each X→Y in Fc do
   if no Rj (1 <= j <=i) contains X,Y
      i←i+1
      Ri ← X U Y
4. If no Rj (1<= j <= i) contains a candidate key for R
      i ← i+1
      Ri ← any candidate key for R
5. return (R1, R2, ..., Ri)

3NF Decomposition Example

Example:
R = (bname, cname, banker, office)
Fc = { banker → bname office,
cname bname → banker}

Q1: candidate keys of R: cname bname or cname banker

Q2: decompose R into 3NF.

Ans: R is not in 3NF: banker → bname office
{bname, office} not a subset of a c. key

3NF: R1 = (banker, bname, office)
R2 = (cname, bname, banker)
R3 = ? Empty (done)

Theory and practice

Performance tuning:

Redundancy not the sole guide to decomposition

Workload matters too!!
- nature of queries run
- mix of updates, queries
- .....

Workload can influence:
BCNF vs 3NF
may further decompose a BCNF into (4NF)
may denormalize (i.e., undo a decomposition or add new columns)