Chapter 3 Design Theory for Relational Databases

Contents

- Functional Dependencies
- Decompositions
- Normal Forms (BCNF, 3NF)
- Multivalued Dependencies (and 4NF)
- Reasoning About FD's + MVD's

Remember our questions:

- Why do we design relations like the example? -- **good design**
- What makes a good relational database schema? -- **no redundancy, no Update/delete anomalies**,
- what we can do if it has flaws? -- **decomposition**

New Question:

- **any standards for a good design?**

  → Normal forms: a condition on a relation schema that will eliminate problems
- **any standards or methods for a decomposition?**

Boyce-Codd Normal Form (BCNF)

- We say a relation *R* is in *BCNF* if whenever *X* ->*Y* is a nontrivial FD that holds in *R*, *X* is a superkey.
  - Remember: *nontrivial* means *Y* is not contained in *X*.
  - Remember, a *superkey* is any superset of a key (not necessarily a proper superset).

Example

Drinkers(<u>name</u>, addr, <u>beersLiked</u>, manf, favBeer)

FD's: name->addr favBeer, beersLiked->manf

- Only key is {name, beersLiked}.
- In each FD, the left side is *not* a superkey.
- *Drinkers* is not in BCNF

Another Example

Beers(name, manf, manfAddr)

FD's: name->manf,   manf->manfAddr

- Only key is {name} .
- name->manf does not violate BCNF, but manf->manfAddr does.
- *Beers* is not in BCNF

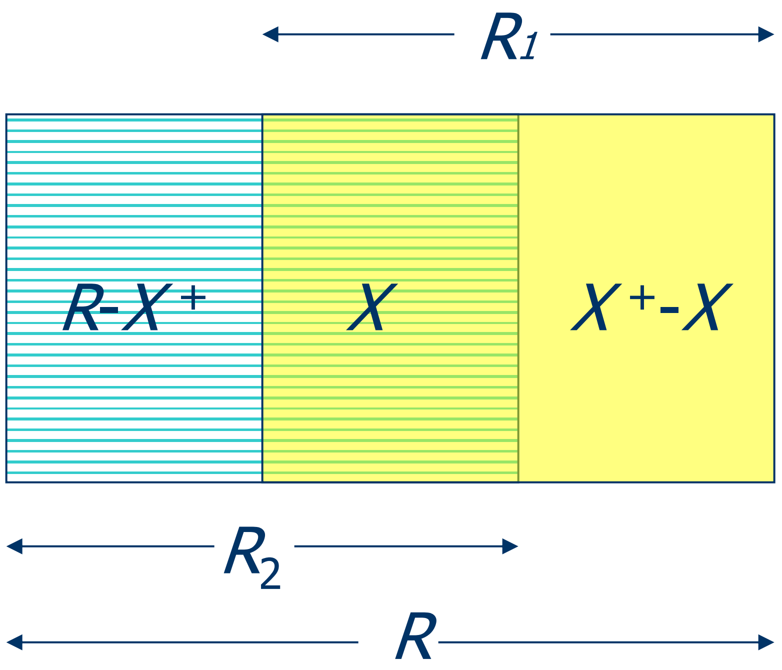
Decomposition into BCNF

- Given: relation *R* with FD's *F*.
- Aim: decompose R to reach BCNF
- Step 1: Look among the given FD's for a BCNF violation *X ->Y*.
  - If any FD following from *F* violates BCNF, then there will surely be an FD in *F* itself that violates BCNF.
- Step 2: Compute X^+.
  - Not all attributes, or else X is a superkey.

Decomposition into BCNF (cont.)

- Step 3: Replace R by relations with schemas:

1. $R_1 = X^+$.
2. $R_2 = R - (X^+ - X)$.



Decomposition into BCNF (cont.)

- Step4*: Project* given FD's *F* onto the two new relations.

given FD's F → find all implied FD's → pick up those FD's which have only attributes needed.

Example: BCNF Decomposition

Drinkers(<u>name</u>, addr, <u>beersLiked</u>, manf, favBeer)

F = name->addr, name -> favBeer, beersLiked->manf

Step1: Pick BCNF violation name->addr.

Step2: Closure the left side: $\{name\}^+ = \{name, addr, favBeer\}$.

Step3: Decomposed relations:

1. Drinkers1(<u>name</u>, addr, favBeer)
2. Drinkers2(<u>name</u>, <u>beersLiked</u>, manf)

Step4: projecting FD's to drinker1 and drinker2

Step5: check drinker 1 and drinker2 for BCNF

Example -- Continued

Given FD's: name->addr, name -> favBeer, beersLiked->manf

All FD's: same as given FD's

- For Drinkers1(name, addr, favBeer), relevant FD's are name->addr and name->favBeer.
  - Thus, {name} is the only key and Drinkers1 is in BCNF.

Example -- Continued

- For Drinkers2(name, beersLiked, manf), the only FD is beersLiked->manf, and the only key is {name, beersLiked}.
  - Violation of BCNF.
- beersLiked^{+} = {beersLiked, manf}, so we decompose *Drinkers2* into:

1. Drinkers3(beersLiked, manf)
2. Drinkers4(name, beersLiked)

Example -- Concluded

- The resulting decomposition of *Drinkers* :
  1. Drinkers1(<u>name</u>, addr, favBeer)
  2. Drinkers3(<u>beersLiked</u>, manf)
  3. Drinkers4(<u>name</u>, <u>beersLiked</u>)
- Notice: *Drinkers1* tells us about drinkers, *Drinkers3* tells us about beers, and *Drinkers4* tells us the relationship between drinkers and the beers they like.

Text book on page 89

Classroom Exercise

- Any two-attribute relation R(A,B) is in BCNF
- True or False ?

Three Cases:

1) No nontrivial FD's
2) $A \rightarrow B$
3) $A \rightarrow B$, $B \rightarrow A$

Third Normal Form -- Motivation

- There is one structure of FD's that causes trouble when we decompose.
- *R(A,B,C)*   *AB* ->*C* and *C* ->*B*.
  - Example: *A* = street address, *B* = city,   *C* = zip code.
- There are two keys, {*A*,*B* } and {*A*,*C* }.
- *C* ->*B* is a BCNF violation, so we must decompose into R1(*AC*), R2(*BC*).

We Cannot Enforce FD's

- Cannot enforce the FD *AB ->C* after decomposition.

Original: R(A,B,C) *AB* ->*C* and *C* ->*B*.

Decompose into

R1(*AC*) *no FD's*

R2(*BC*) *with C->B*

- Assume *A* = street, *B* = city, and *C* = zip

A = street, B = city,
and C = zip

We Cannot Enforce FD's (cont.)

C→B
keeps

A	C
545 Tech Sq.	02138
545 Tech Sq.	02139

B	C
Cambridge	02138
Cambridge	02139

Join tuples with equal C (zip codes).

A	B	C
545 Tech Sq.	Cambridge	02138
545 Tech Sq.	Cambridge	02139

FD: A (street) B(city) -> C(zip) is violated
by the database as a whole.

3NF Let's Us Avoid This Problem

- 3rd Normal Form (3NF) modifies the BCNF condition so we do not have to decompose in this problem situation.
- An attribute is *prime* if it is a member of any key.
- *X ->A* violates 3NF if and only if *X* is not a superkey, and also *A* is not prime.

Example: 3NF

- In our problem situation **R(A,B,C)** with FD's *AB ->C* and *C ->B*, we have keys *AB* and *AC*.
- Thus *A*, *B*, and *C* are each prime.
- Although *C ->B* violates BCNF, it does not violate 3NF.

- **R(A,B,C)** above is in 3NF, not in BCNF

BCNF vs. 3NF

	conditions	example
BCNF	If *X* ->*Y* is a nontrivial FD that holds in *R*, *X* is a superkey.	R(A,B,C) with A→B, A→C
3NF	If *X* ->*Y* is a nontrivial FD that holds in *R*, *X* is a superkey, or Y is a prime	R(A,B,C) with *AB* ->*C* and *C* ->*B*.

2 NF: no nonkey attribute is dependent on only a portion of the primary key. R(A,B,C) with A→B, B→C

1 NF: every component of every tuple is an atomic value.

Properties of a decomposition

- Elimination of anomalies by a decomposition, it needs other two properties:

1. *Lossless Join* : it should be possible to project the original relations onto the decomposed schema, and then reconstruct the original.
2. *Dependency Preservation* : it should be possible to check in the projected relations whether all the given FD's are satisfied.

Decomposition for 3NF and BCNF

- We can get (1: *Lossless Join*) with a BCNF decomposition.
- we can't always get (1) and (2 : *Dependency Preservation*) with a BCNF decomposition.
- We can get both (1) and (2) with a 3NF decomposition.

Testing for a Lossless Join

- If we project R onto R_1, R_2,..., R_k , can we recover R by rejoining?
- → Any tuple in R can be recovered from its projected fragments.
- → So the only question is: when we rejoin, do we ever get back something we didn't have originally?

Example

- Any tuple in *R* can be recovered from its projected fragments.
- Not less, not more.

As long as FD b→c holds, the joining of two projected tuples cannot produce a bogus tuple

A	B	C
1	2	3
4	2	5

A	B
1	2
4	2

$\bowtie$

B	C
2	3
2	5

A	B	C
1	2	3
1	2	5
4	2	3
4	2	5

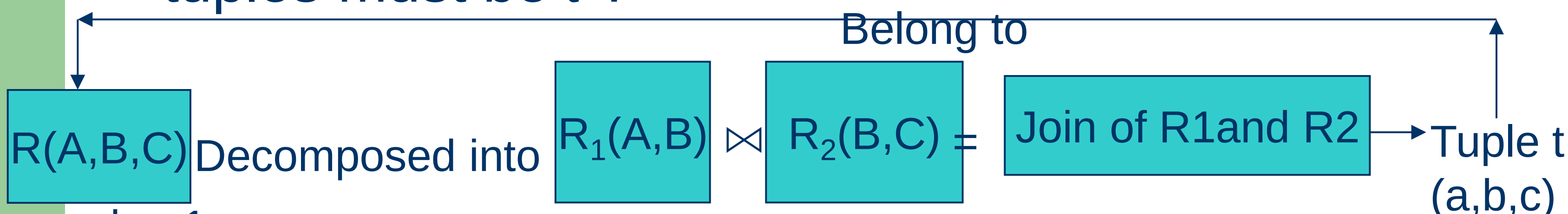
textbook on page 98

The Chase Test (an example)

Aim: to prove that a tuple t in the join, using FD's in F, also to be a tuple in R.

Method: Suppose tuple *t* comes back in the join.

- Then *t* is the join of projections of some tuples of *R*, one for each R_i of the decomposition.
- Can we use the given FD's to show that one of these tuples must be *t* ?



a b c1

Because B→ C, c1= c , therefore a,b,c is in R

a2 b c

The Chase Test – (method)

1. Start by assuming $t = abc$... .
2. For each i, there is a tuple s_i of R that has a, b, c,... in the attributes of R_i.
3. s_i can have any values in other attributes.
4. We'll use the same letter as in t, but with a subscript, for these components.

Example: The Chase

- Let *R* = *ABCD*, the given FD's be *C->D* and *B ->A*
- Suppose the decomposition be *AB*, *BC*, and *CD*.
- Question: **Is it a lossless join or not?**

R(ABCD) == R1(AB) ⋈ R2(BC) ⋈ R3(CD)

Example: The Chase (cont.)

1. Suppose the tuple t = abcd is the join of tuples projected onto *AB*, *BC*, *CD*.
2. For each i, there is a tuple s_i of R that has a, b from R1(AB), b,c from R2(BC) and c,d from R3(CD)
3. $a_i\ b_i\ c_i\ d_i$ are any values

Aim: t=abcd is also in the R

The *Tableau*

A	B	C	D
a	b	c_1	d_1
$\cancel{a_2}\ a$	b	c	$\cancel{d_2}\ d$
a_3	b_3	c	d

Use B->A

Use C->D

We've proved the second tuple must be t.

Summary of the Chase Test method

1. If two rows agree in the left side of a FD, make their right sides agree too.
2. Always replace a subscripted symbol by the corresponding unsubscripted one, if possible.
3. If we ever get an unsubscripted row, we know any tuple in the project-join is in the original (the join is lossless).
4. Otherwise, the final tableau is a counterexample.

Example: Lossy Join (more tuples)

- Same relation $R = ABCD$ and same decomposition: AB, BC, and CD.
- But with only the FD $C \rightarrow D$.

These projections rejoin to form *abcd*

The *Tableau*

A	*B*	*C*	*D*
a	*b*	c_1	d_1
a_2	*b*	*c*	$\cancel{d_2}$ *d*
a_3	b_3	*c*	*d*

Use *C->D*

These three tuples are an example *R* that shows the join lossy. *abcd* is not in *R.* More tuples

Some results

- Some decompositions can not keep lossless join (lossy join).
- Use **chase method** to find out whether the decomposition is lossy join.
- **BCNF decomposition is lossless join**, sometimes it can not keep functional dependencies.
- Relations with 3NF keep lossless join and also functional dependencies.
- How to decompose relations to reach 3NF?

3NF Synthesis Algorithm

- We can always construct a decomposition into 3NF relations with a lossless join and dependency preservation.
- Need *minimal basis* for the FD's:

1. Right sides are single attributes.
2. No FD can be removed.
3. No attribute can be removed from a left side.

Constructing a Minimal Basis

1. Split right sides.
2. Repeatedly try to remove an FD and see if the remaining FD's are equivalent to the original.
3. Repeatedly try to remove an attribute from a left side and see if the resulting FD's are equivalent to the original.

3NF Synthesis – method

1. Find a minimal basis for F
2. One relation for each FD in the minimal basis.
    1. Schema is the union of the left and right sides.
    2. $X \rightarrow A$ then (XA) is a schema.
3. If no key is contained in an FD, then add one relation whose schema is some key.

Algorithm 3.26 is on pp.103

Example: 3NF Synthesis

- Relation R = ABCD.
- FD's *A->B* and *A->C*. Key is AD
- Decomposition: AB and AC from the FD's, plus AD for a key.

R is decomposed into R1(AB), R2(AC), R3(AD)

Another example

- Relation R(A,B,C,D,E) and FD's AB→C, C→B,A→D

Using 3NF synthesis to decompose:

1) A minimal basis
2) R1(ABC) R2(CB) R3(AD)
3) R2 is a part of R1, delete R2
4) No key is in R1, R3, add a key R4(ABE)

R has two keys: ABE, ACE. Add one of them

Classroom exercises

Given R(A,B,C,D,E) FD's AB→C, C→B, A→D

- To test R1(ABC), R3(AD), R4(ABE) is in 3NF
- To test whether functional dependency keeps in the R1,R3,R4
- To test the decomposition is lossless.

Why It Works (3NF synthesis)

- Preserves dependencies: each FD from a minimal basis is contained in a relation, thus preserved.
- Lossless Join: use the chase to show that the row for the relation that contains a key can be made all-unsubscripted variables.

Question:

Why we say "BCNF decomposition" ,"3NF synthesis"?

BCNF decomposition algorithm

Input: relation R + FDs for R

Output: decomposition of R into BCNF relations

With “lossless join”

1. Compute keys for R
2. Repeat until all relations are in BCNF:

Pick any R’ with A→ B that violates BCNF

Decompose R’ into R1(A+) and R2(A, rest)

Compute FDs for R1 and R2

Compute keys for R1 and R2

3NF synthesis

- Input: relation R + FDs for R
- Output: decomposition of R into 3NF

With “lossless join” and keep FD’s

1. Computer key of R
2. Find a minimal basis for F
3. One relation for each FD in the minimal basis.
4. If no key is contained in an FD, then add one relation whose schema is some key.

Summary

- Conditions of Norm Forms (BCNF, 3NF)
- The way to decompose in order to reach BCNF
- The way to decompose in order to reach 3NF
- The way to test the join is lossless join