Databases
Functional dependencies and normalization

Hans Philippi

November 21, 2017

Functional dependencies

The database design problem

- Suppose we have a db schema: is it *good*?
- Suppose we have a db schema derived from an ER diagram: is it *good*?
- A db schema seems to be good if it helps us to avoid redundancy and inconsistency, but are there more quality issues?
- When is a db schema *good* anyway?

normalization theory

Functional dependencies

Goals:

- define precise notions of the qualities of a relational database schema
- define algorithms to generate 'good' relational database schemes

Functional dependencies

The *Holy Grail* of normalization theory

`input:`

- the set of all relevant attributes
- a set of constraints based on the attribute semantics

`output:`

- an 'optimal' relational schema for these attributes

Functional dependencies: notational conventions

A set of attributes is represented without the brackets { } and the comma separator from set theory.

Concatenation of symbols denotes the union operator.

An example: instead of

$$X = \{A, B, C\}$$
$$Y = \{C, D, E\}$$
$$X \cup Y = \{A, B, C, D, E\}$$

we will write

$$X = ABC$$
$$Y = CDE$$
$$XY = ABCDE$$

Letters from the beginning of the alphabet denote attributes, letters from the end denote sets of attributes.

Functional dependencies

What can go wrong when we split a table?

r			
A	**B**	**C**	**D**
a1	b1	c1	d1
a2	b2	c1	d2

We split r using projections ...

r_1	
A	**B**
a1	b1
a2	b2

r_2		
A	**C**	**D**
a1	c1	d1
a2	c1	d2

Functional dependencies

... and we reconstruct r using the natural join. Great ...

r			
A	**B**	**C**	**D**
a1	b1	c1	d1
a2	b2	c1	d2

r_1	
A	**B**
a1	b1
a2	b2

r_2		
A	**C**	**D**
a1	c1	d1
a2	c1	d2

$r_1 \bowtie r_2$			
A	**B**	**C**	**D**
a1	b1	c1	d1
a2	b2	c1	d2

Functional dependencies

… but, uh …

r			
A	**B**	**C**	**D**
a1	b1	c1	d1
a2	b2	c1	d2

r_3		
A	**B**	**C**
a1	b1	c1
a2	b2	c1

r_4	
C	**D**
c1	d1
c1	d2

$r_3 \bowtie r_4$			
A	**B**	**C**	**D**
a1	b1	c1	d1
a1	b1	c1	d2
a2	b2	c1	d1
a2	b2	c1	d2

Functional dependencies

After splitting and reconstructing, we might see *spurious tuples*

r			
A	**B**	**C**	**D**
a1	b1	c1	d1
a2	b2	c1	d2

r_3		
A	**B**	**C**
a1	b1	c1
a2	b2	c1

r_4	
C	**D**
c1	d1
c1	d2

$r_3 \bowtie r_4$			
A	**B**	**C**	**D**
a1	b1	c1	d1
a1	b1	c1	d2
a2	b2	c1	d1
a2	b2	c1	d2

Functional dependencies

Definition:
Suppose we have a relation schema R (with constraints).

A <u>decomposition</u> of R is a set of relation schemas $R_1, \ldots, R_n$ such that
(i) each R_i consists of attributes in R and
(ii) each attribute of R occurs in at least one R_j

Functional dependencies

Definition:
Suppose we have a relation schema R (with constraints).
A decomposition $R_1, R_2, \ldots, R_n$ of R is called
<u>lossless</u> iff for each valid relation $r(R)$:

$$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \ldots \bowtie \pi_{R_n}(r)$$

Requirement 1:
Your decomposition should be *lossless*

Functional dependencies

A crucial notion in the field of database design is the *functional depencency (FD)*

- FDs are omnipresent when expressing constraints on database schemas
- FDs are essential for a deeper understanding of data redundancy
- FDs are the key to solving the lossless decomposition problem
- an FD states that the value of an attribute is fully determined by one or more other attributes

Functional dependencies: examples

Within a table

Address (street, number, zipcode, city)

we see that

zipcode $\rightarrow$ street (zipcode determines street)

...

Functional dependencies: examples

Within a table

Address (street, number, zipcode, city)

we see that

zipcode $\rightarrow$ street (zipcode determines street)

zipcode $\rightarrow$ city

street, number, city $\rightarrow$ zipcode

Functional dependencies

Definition:
Suppose we have relation r with schema R.
Suppose $X, Y \subseteq attr(R)$.
On r, the
functional dependency (FD) $X \to Y$ holds
iff for each pair of tuples $t_1, t_2 \in r$:
$t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$.

Note:
say X *determines* Y, not X *implies* Y !

Functional dependencies

FindTheFDs			
A	**B**	**C**	**D**
2	4	6	8
1	3	5	7
2	4	6	7
3	4	6	6
1	2	6	6
1	3	5	9

Does $A \rightarrow B$ hold?
Does $B \rightarrow C$ hold?
Does $C \rightarrow B$ hold?
Does $AB \rightarrow C$ hold?

Functional dependencies & lossless decompositions

Theorem:
Suppose we have a relational schema $R(XYZ)$.
The following holds:

$$\text{If } X \rightarrow Y \text{ then}$$
$$\text{the decomposition } R_1(XY), R_2(XZ) \text{ is lossless}$$

Functional dependencies may cause redundancy

redundancy: case 1

CATALOGUE			
brand	**type**	**price**	**distributor**
Batavus	Flits	1500	Batavus NL
Batavus	Opoe	500	Batavus NL
Trek	Downhill	4800	Enterprise
Trek	Earthquake	7500	Enterprise
Cannondale	Jumper	3900	C'Europe
Cannondale	Downhill	5600	C'Europe
Specialized	Jumper	3500	Tech Unlimited

Functional dependencies may cause redundancy

redundancy: case 1

CATALOGUE			
brand	**type**	**price**	**distributor**
Batavus	Flits	1500	Batavus NL
Batavus	Opoe	500	Batavus NL
Trek	Downhill	4800	Enterprise
Trek	Earthquake	7500	Enterprise
Cannondale	SmartJumper	3900	C'Europe
Cannondale	Downhill	5600	C'Europe
Specialized	StumpJumper	3500	Tech Unlimited

Redundancy caused by FD brand $\rightarrow$ distributor
But... this FD also guarantees a lossless decomposition

FDs: lossless decomposition avoids redundancy

DISTRIBUTION	
brand	**distributor**
Batavus	Batavus NL
Trek	Enterprise
Cannondale	C'Europe
Specialized	Tech Unlimited

CATALOGUE		
brand	**type**	**price**
Batavus	Flits	1500
Batavus	Opoe	500
Trek	Downhill	4800
Trek	Earthquake	7500
Cannondale	SmartJumper	3900
Cannondale	Downhill	5600
Specialized	StumpJumper	3500

Functional dependencies may cause redundancy

redundancy: case 2

RISC		
city	**region**	**rise**
Amsterdam	Randstad	40 %
Utrecht	Randstad	40 %
Rotterdam	Randstad	40 %
Maastricht	Limburg	10 %
Den Haag	Randstad	40 %
Almelo	Twente	25 %
Hengelo	Twente	25 %
Oldenzaal	Twente	25 %
Veenendaal	Gelderse Vallei	0 %

Functional dependencies may cause redundancy

redundancy: case 2

RISC		
city	**region**	**rise**
Amsterdam	Randstad	40 %
Utrecht	Randstad	40 %
Rotterdam	Randstad	40 %
Maastricht	Limburg	10 %
Den Haag	Randstad	40 %
Almelo	Twente	25 %
Hengelo	Twente	25 %
Oldenzaal	Twente	25 %
Veenendaal	Gelderse Vallei	0 %

FDs: lossless decomposition avoids redundancy

CITY-REGION	
city	**region**
Amsterdam	Randstad
Utrecht	Randstad
Rotterdam	Randstad
Maastricht	Limburg
Den Haag	Randstad
Almelo	Twente
Hengelo	Twente
Oldenzaal	Twente
Veenendaal	Gelderse Vallei

REGION-RISE	
region	**rise**
Randstad	40 %
Limburg	10 %
Twente	25 %
Gelderse Vallei	0 %

Functional dependencies

Requirement 2:
Your decomposition should avoid *redundancy*

Making this statement more precise requires a profound study of FDs and their structure

Functional dependencies

Suppose we have a Universe $R(A, B, C, D)$, investigated by two teams

Team 1 finds:

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

Team 2 finds:

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

Functional dependencies

Suppose we have a Universe $R(A, B, C, D)$, investigated by three teams

Team 1 finds:

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

Team 2 finds:

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

Team 3 finds:

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

$$A \rightarrow A, AB \rightarrow A, AB \rightarrow C, \ldots, ABCD \rightarrow ABCD$$

Functional dependencies: inference

Inference rules for FDs:

(IR_1) $Y \subseteq X \Rightarrow X \rightarrow Y$ (reflexivity)
(IR_2) $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$ (augmentation)
(IR_3) $X \rightarrow Y,\ Y \rightarrow Z \Rightarrow X \rightarrow Z$ (transitivity)

Rules $IR_1 - IR_3$ are known as the
Armstrong axioms
these rules are *sound* and *complete*

INTERMEZZO

Are the following rules valid?

- $X \rightarrow Y, X \rightarrow Z \Rightarrow X \rightarrow YZ$
- $XY \rightarrow Z \Rightarrow X \rightarrow Z, Y \rightarrow Z$
- $X \rightarrow YZ \Rightarrow X \rightarrow Y$

INTERMEZZO

Are the following rules valid?

- $X \to Y, X \to Z \Rightarrow X \to YZ$ Yes
- $XY \to Z \Rightarrow X \to Z, Y \to Z$ No
- $X \to YZ \Rightarrow X \to Y$ Yes

Functional dependencies: calculations

In general, it is important to determine *all* the attributes depending on a left side X

Definition:
Suppose we have a schema R and a set FD's F on R.
The closure (X-closure) X^+ of an attribute set X
is the set of all attributes $A \in R$
such that $X \rightarrow A$ holds.

Functional dependencies: X-closure algorithm

Algorithm
INPUT: $X \subseteq R, F$
OUTPUT: X^+
METHOD:

```
X+ = X;
repeat
    oldX+ = X+;
    for each FD U → V in F {
        if U ⊆ X+ then X+ = X+ ∪ V;
    }
until oldX+ == X+
```

INTERMEZZO

We have a schema attributes $R = (ABCDEFG)$ and a set FDs $\mathbf{F} = \{C \rightarrow DE, A \rightarrow C, G \rightarrow D, B \rightarrow G\}$

Determine $A^+, B^+, C^+, (AG)^+$

Give also a key for R.

INTERMEZZO

We have a schema attributes $R = (ABCDEFG)$ and a set FDs $\mathbf{F} = \{C \rightarrow DE, A \rightarrow C, G \rightarrow D, B \rightarrow G\}$

- $A^+ = A$ ...
- $A^+ = AC$ ...
- $A^+ = ACDE$

INTERMEZZO

We have a schema attributes $R = (ABCDEFG)$ and a set FDs
$\mathbf{F} = \{C \rightarrow DE, A \rightarrow C, G \rightarrow D, B \rightarrow G\}$

- $A^+ = ACDE$
- $B^+ = BGD$
- $C^+ = CDE$
- $(AG)^+ = AGCDE$

Note that: $(AB)^+ = ABCGDE$
Key: ABF

Functional dependencies

Formalization of *avoiding redundancy:*

Definition:
Suppose we have a schema R and a set FDs F.
R is in <u>BCNF</u> (w.r.t. F) iff each left side of a non-trivial FD is a superkey

BCNF: Boyce-Codd Normal Form

Functional dependencies

Algorithm
INPUT: a schema R, a set of FDs F
OUTPUT:
 a lossless BCNF-decomposition of R
METHOD:
 while there is a schema S not in BCNF $\{$
  suppose the villain is $X \to Y$;
  let Z be the set of remaining attributes in S;
  split $S(XYZ)$ into $S_1(XY), S_2(XZ)$
 $\}$

INTERMEZZO

We have a relation schema $R(ABCDE)$
and a set of fd's
$F = \{A \rightarrow BC, C \rightarrow D, D \rightarrow E\}$

Give at least two BCNF decompositions

Do you have a preference for one of the decompositions?

INTERMEZZO

We have a relation schema $R(ABCDE)$
and a set of fd's
$F = \{A \to BC, C \to D, D \to E\}$

Decomposition 1: first split using $C \to D$, ...

$$(CD), (ABCE)$$

..., then split $(ABCE)$ using $C \to E$ (!)

$$(CD), (CE), (ABC)$$

INTERMEZZO

We have a relation schema $R(ABCDE)$
and a set of fd's
$F = \{A \to BC, C \to D, D \to E\}$

Decomposition 1: first split using $C \to D$, then using $C \to E$

$$(CD), (CE), (ABC)$$

Decomposition 2: first split using $D \to E$, then using $C \to D$

$$(CD), (DE), (ABC)$$

Functional dependencies

Algorithm (refined version)
INPUT: a schema R, a set of FDs F
OUTPUT:
    a lossless BCNF-decomposition of R
METHOD:
    while there is a schema S not in BCNF $\{$
        suppose the villain has left side X;
        let $Y = X^+$ without X;
        let Z be the set of remaining attributes in S;
        split $S(XYZ)$ into $S_1(XY), S_2(XZ)$
    $\}$

INTERMEZZO

We have a relation schema $R(ABCDE)$
and a set of fd's
$F = \{A \to BC, C \to D, D \to E\}$

Decomposition 1: first split using $C \to DE$, (i.e. C^+) ...

$$(CDE), (ABC)$$

..., then split (CDE) using $D \to E$

$$(CD), (DE), (ABC)$$