Functional Dependencies and Relational Schema Design

Friday, October 12 2001

Outline

- Functional dependencies and keys (3.5)
- Normal forms: BCNF (3.7)

Functional Dependencies

- A form of constraint (hence, part of the schema)
- Finding them is part of the database design
- Also used in normalizing the relations

Functional Dependencies

Definition:

If two tuples agree on the attributes

$A_1, A_2, \ldots A_n$

then they must also agree on the attributes

$B_1, B_2, \ldots B_m$

Formally: $A_1, A_2, \ldots A_n \longrightarrow B_1, B_2, \ldots B_m$

Main (and simplest) example: keys

Examples

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E1847	John	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	lawyer

- EmpID $\longrightarrow$ Name, Phone, Position
- Position $\longrightarrow$ Phone
- but Phone $\not\longrightarrow$ Position

In General

- To check A $\longrightarrow$ B, erase all other columns

…	A	…	B	
	X1		Y1	
	X2		Y2	
	…		…	

- check if the remaining relation is many-one (called ***functional*** in mathematics)

Example

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E1847	John	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	lawyer

More Examples

Product: name → price, manufacturer
Person: ssn → name, age
Company: name → stock price, president

Key of a relation is a set of attributes that:

- functionally determines all the attributes of the relation
- none of its subsets determines all the attributes.

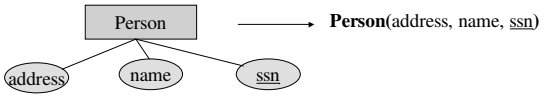
Superkey: a set of attributes that contains a key.

Finding the Keys of a Relation

Given a relation constructed from an E/R diagram, what is its key?

Rules:

1. If the relation comes from an entity set,
   the key of the relation is the set of attributes which is the key of the entity set.

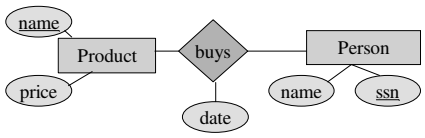


Person(address, name, <u>ssn</u>)

Finding the Keys

Rules:

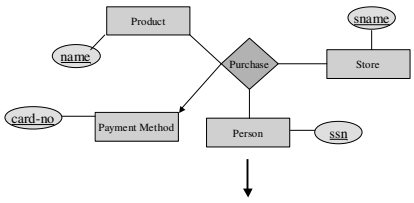
2. If the relation comes from a many-many relationship,
   the key of the relation is the set of all attribute keys in the relations corresponding to the entity sets



buys(<u>name, ssn</u>, date)

Finding the Keys

Except: if there is an arrow from the relationship to E, then we don't need the key of E as part of the relation key.



Purchase(<u>name , sname, ssn</u>, card-no)

Finding the Keys

More rules:

- Many-one, one-many, one-one relationships
- Multi-way relationships
- Weak entity sets

(Try to find them yourself, check book)

Rules for FD's

$A_1, A_2, \ldots A_n \longrightarrow B_1, B_2, \ldots B_m$

Splitting rule and Combing rule

Is equivalent to

$A_1, A_2, \ldots A_n \longrightarrow B_1$

$A_1, A_2, \ldots A_n \longrightarrow B_2$

…

$A_1, A_2, \ldots A_n \longrightarrow B_m$

Rules in FD's (continued)

$A_1, A_2, \ldots A_n \longrightarrow A_i$ **Trivial Rule**

Why ?

Rules in FD's (continued)

Transitive Closure Rule

If $A_1, A_2, \ldots A_n \longrightarrow B_1, B_2 \ldots, B_m$

and $B_1, B_2, \ldots B_m \longrightarrow C_1, C_2 \ldots, C_p$

then $A_1, A_2, \ldots A_n \longrightarrow C_1, C_2 \ldots, C_p$

Why ?

Closure of a set of Attributes

Given a set of attributes ***{A1, …, An}*** and a set of dependencies S.
Problem: find all attributes *B* such that:
any relation which satisfies S also satisfies:
A1, …, An → B

The **closure** of *{A1, …, An}*, denoted *{A1, …, An}* $^+$,
is the set of all such attributes *B*

Closure Algorithm

Start with X={A1, …, An}.

Repeat until X doesn't change **do**:

if $B_1, B_2, \ldots B_n \longrightarrow C$ is in S, **and**
$B_1, B_2, \ldots B_n$ are all in X, **and**
C is not in X

then

add C to X.

Example

A B → C
A D → E
B → D
A F → B

Closure of {A,B}: X = {A, B, }

Closure of {A, F}: X = {A, F, }

Why Is the Algorithm Correct ?

- Show the following by induction:
  - For every B in X:
    - $A1, ..., An \rightarrow B$
- Initially $X = \{A1, ..., An\}$ -- holds
- Induction step: $B1, ..., Bm$ in X
  - Implies $A1, ..., An \rightarrow B1, ..., Bm$
  - We also have $B1, ..., Bm \rightarrow C$
  - By transitivity we have $A1, ..., An \rightarrow C$
- This shows that the algorithm is *sound*; need to show it is *complete*

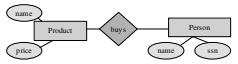
Relational Schema Design (or Logical Design)

Main idea:

- Start with some relational schema
- Find out its FD's
- Use them to design a better relational schema

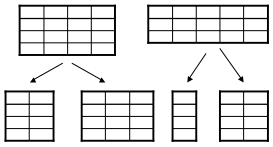
Relational Schema Design

Conceptual Model:



Relational Model: (plus FD's)

Normalization:



Relational Schema Design

Goal: eliminate anomalies

- Redundancy anomalies
- Deletion anomalies
- Update anomalies

Relational Schema Design

Recall set attributes (persons with several phones):

Name	SSN	Phone Number
Fred	123-321-99	(201) 555-1234
Fred	123-321-99	(206) 572-4312
Joe	909-438-44	(908) 464-0028
Joe	909-438-44	(212) 555-4000

Note: SSN no longer a key here

Anomalies:

Redundancy = repeat data

update anomalies = need to update in many places

deletion anomalies = need to delete many tuples

Relation Decomposition

Break the relation into two:

SSN	Name
123-321-99	Fred
909-438-44	Joe

SSN	Phone Number
123-321-99	(201) 555-1234
123-321-99	(206) 572-4312
909-438-44	(908) 464-0028
909-438-44	(212) 555-4000

Decompositions in General

Let *R* be a relation with attributes $A_1, A_2, \ldots A_n$

Create two relations *R1* and *R2* with attributes

$B_1, B_2, \ldots B_m$ $C_1, C_2, \ldots C_1$

Such that:

$B_1, B_2, \ldots B_m \cup C_1, C_2, \ldots C_1 = A_1, A_2, \ldots A_n$

And

-- *R1* is the projection of *R* on $B_1, B_2, \ldots B_m$

-- *R2* is the projection of *R* on $C_1, C_2, \ldots C_1$

Incorrect Decomposition

- Sometimes it is incorrect:

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
DoubleClick	29.99	Camera

Decompose on : **Name, Category** and **Price, Category**

Incorrect Decomposition

Name	Category
Gizmo	Gadget
OneClick	Camera
DoubleClick	Camera

Price	Category
19.99	Gadget
24.99	Camera
29.99	Camera

When we put it back:

Cannot recover information

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
OneClick	29.99	Camera
DoubleClick	24.99	Camera
DoubleClick	29.99	Camera

Normal Forms

First Normal Form = all attributes are atomic

Second Normal Form (2NF) = old and obsolete

Third Normal Form (3NF) = this lecture

Boyce Codd Normal Form (BCNF) = this lecture

Others...

Boyce-Codd Normal Form

A simple condition for removing anomalies from relations:

A relation R is in BCNF if and only if:

Whenever there is a nontrivial dependency $A_1, A_2, \ldots A_n \rightarrow B$ for R , it is the case that $\{ A_1, A_2, \ldots A_n \}$ a super-key for R.

In English (though a bit vague):

Whenever a set of attributes of *R* is determining another attribute, should determine ***all*** the attributes of *R*.

Example

Name	SSN	Phone Number
Fred	123-321-99	(201) 555-1234
Fred	123-321-99	(206) 572-4312
Joe	909-438-44	(908) 464-0028
Joe	909-438-44	(212) 555-4000

What are the dependencies?

SSN→Name

What are the keys?

Name, SSN, PhoneNumber

Is it in BCNF?

Decompose it into BCNF

SSN	Name
123-321-99	Fred
909-438-44	Joe

SSN → Name

SSN	Phone Number
123-321-99	(201) 555-1234
123-321-99	(206) 572-4312
909-438-44	(908) 464-0028
909-438-44	(212) 555-4000

SSN → PhoneNumber

What About This?

Name	Price	Category
Gizmo	$19.99	gadgets
OneClick	$24.99	camera

Name → Price, Category

It is already in BCNF

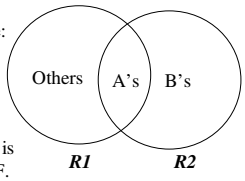
BCNF Decomposition

Find a dependency that violates the BCNF condition:

$A_1, A_2, \ldots A_n \longrightarrow B_1, B_2, \ldots B_m$

Heuristics: choose $B_1, B_2, \ldots B_m$ "as large as possible"

Decompose:



Find a 2-attribute relation that is not in BCNF.

R1 ***R2***

Continue until there are no BCNF violations left.

Example Decomposition

Person:

Name	SSN	Age	EyeColor	PhoneNumber

Functional dependencies:

SSN → Name, Age, Eye Color

BNCF: Person1(SSN, Name, Age, EyeColor),
Person2(SSN, PhoneNumber)

What if we also had an attribute Draft-worthy, and the FD:

Age → Draft-worthy

Other Example

- R(A,B,C,D) A → B, B → C
- Key: A, D
- Violations of BCNF: A → B, A → C, A → BC
- Pick A → BC: split into R1(A,BC) R2(A,D)
- What happens if we pick A → B first ?

Correct Decompositions

A decomposition is *lossless* if we can recover:

R(A,B,C)

R1(A,B) R2(A,C)

R'(A,B,C) = R(A,B,C)

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