CIS 560 – Database System Concepts

Lecture 9

Database Design: Functional Dependencies

September 16, 2013

Credits for slides: Suciu, Chang, Ullman.

Outline

Last time:

- DB Design: E/R Diagrams (Sections 4.1-4.5)

Today:

- DB Design: Functional Dependencies (3.1 – 3.2)

Next:

- DB Design: Normalization (3.3-3.4)
- Transactions in SQL

2

Review

- ISA relationships
- Weak entity sets

Database Design

- Why do we need it?
  - Agree on structure of the database before deciding on a particular implementation.
- Consider issues such as:
  - What entities to model
  - How entities are related
  - What constraints exist in the domain
  - How to achieve ***good*** designs

Where We Are in Terms of DB Design

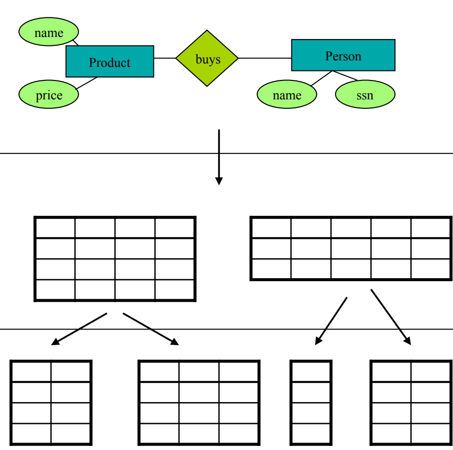
- We have designed an E/R diagram, and translated it into a relational DB schema R = set of R1, R2, ...
- We know how to do the following
  - Specify relevant constraints over R
  - implement R in SQL
  - start using it, making sure the constraints always remain valid
- However, R may not be well-designed, thus causing us a lot of problems

Relational Schema Design

Conceptual Model:

Relational Model:
plus constraints

Normalization:
Eliminates ***anomalies***



Is This Good Design?

Student

Name	GPA	Courses
Alice	3.8	Math DB OS
Bob	3.7	DB OS
Carol	3.9	Math OS

First Normal Form (1NF)

- A database schema is in First Normal Form if all tables are flat.

Student

Name	GPA	Courses
Alice	3.8	Math DB OS
Bob	3.7	DB OS
Carol	3.9	Math OS

May need to add keys

Student

Name	GPA
Alice	3.8
Bob	3.7
Carol	3.9

Takes

Student	Course
Alice	Math
Carol	Math
Alice	DB
Bob	DB
Alice	OS
Carol	OS

Course

Course
Math
DB
OS

Is This Good Design?

Set attributes (persons with several phones):

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Topeka
Fred	123-45-6789	206-555-6543	Topeka
Joe	987-65-4321	908-555-2121	Wichita

One person may have multiple phones, but lives in only one city

Anomalies:

- Redundancy = repeat data
- Update anomalies = Fred moves to "Manhattan"
- Deletion anomalies = Joe deletes his phone number: what is his city ?

Data Anomalies

When a database is poorly designed we get anomalies:

<u>Redundancy</u>: data is repeated

<u>Updated anomalies</u>: need to change in several places

<u>Delete anomalies</u>: may lose data when we don't want

How can we fix these problems?

Relation Decomposition

Break the relation into two:

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Topeka
Fred	123-45-6789	206-555-6543	Topeka
Joe	987-65-4321	908-555-2121	Wichita

Name	SSN	City
Fred	123-45-6789	Topeka
Joe	987-65-4321	Wichita

SSN	PhoneNumber
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121

Anomalies have gone:

- No more repeated data
- Easy to move Fred to "Manhattan" (how?)
- Easy to delete all Joe's phone number (how?)

How Do We Obtain a Good Design?

- Start with the original DB schema R
- Transform it until we get a good design R*
- Desirable properties for R*
  - must preserve the information of R
  - must have minimal amount of redundancy
  - must be dependency-preserving
    - if R is associated with a set of constraints C, then it should be easy to also check C over R*
  - (must also give good query performance)

OK, but ...

- How do we recognize a good design R*?
- How do we transform R into R*?

- What we need is the “theory” of …

Normal Forms

- DB gurus have developed many normal forms
- Most important ones
  - Boyce-Codd, 3rd, and 4th normal forms
- If R* is in one of these forms, then R* is guaranteed to achieve certain good properties
  - e.g., if R* is in Boyce-Codd NF, it is guaranteed to not have certain types of redundancy
- DB gurus have also developed algorithms to transform R into R* that is in some of these normal forms

Normal Forms (cont.)

- DB gurus have also discussed trade-offs among normal forms
- Thus, all we have to do is
  - learn these normal forms
  - transform R into R* in one of these forms
  - carefully evaluate the trade-offs
- Many of these normal forms are defined based on various constraints
  - functional dependencies and keys

Relational Schema Design (or Logical Design)

Main idea:

- Start with some relational schema
- Find out its ***functional dependencies***
- Use them to design a better relational schema

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

$$\boxed{A_1, A_2, \ldots, A_n}$$

then they must also agree on the attributes

$$\boxed{B_1, B_2, \ldots, B_m}$$

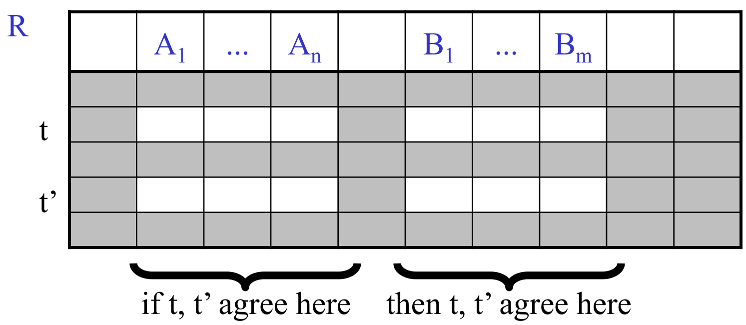
Formally:

$$\boxed{A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m}$$

When Does a Functional Dependency Hold

Definition: $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$ holds in R if:

$\forall t, t' \in R, (t.A_1 = t'.A_1 \wedge \ldots \wedge t.A_n = t'.A_n \Rightarrow t.B_1 = t'.B_1 \wedge \ldots \wedge t.B_m = t'.B_m)$



Examples

An FD holds, or does not hold on an instance:

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

EmpID $\rightarrow$ Name, Phone, Position

Position $\rightarrow$ Phone

but not Phone $\rightarrow$ Position

Example

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876 ←	Salesrep
E1111	Smith	9876 ←	Salesrep
E9999	Mary	1234	Lawyer

Position → Phone

Example

EmpID	Name	Phone	Position
E0045	Smith	1234 →	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234 →	Lawyer

but not Phone → Position

Example

FD's are constraints:
- On some instances they hold
- On others they don't

name → color
category → department
color, category → price

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Green	Toys	99

Does this instance satisfy all the FDs?

Example

name → color
category → department
color, category → price

name	category	color	department	price
Gizmo	Gadget	Green	Toys	49
Tweaker	Gadget	Black	Toys	99
Gizmo	Stationary	Green	Office-supp.	59

What about this one?

An Interesting Observation

If all these FDs are true:

name → color
category → department
color, category → price

Then this FD also holds:

name, category → price

Why?

Goal: Find ALL Functional Dependencies

Anomalies occur when certain “bad” FDs hold

- We know some of the FDs
- Need to find *all* FDs, then look for the bad ones
- How can we find all FDs?
  - Armstrong’s rules

Armstrong's Rules (1/3)

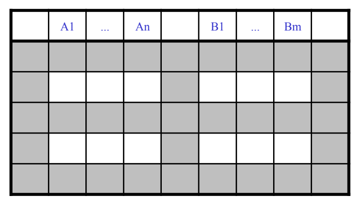
$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$

Is equivalent to

$A_1, A_2, \ldots, A_n \rightarrow B_1$
$A_1, A_2, \ldots, A_n \rightarrow B_2$
. . . . .
$A_1, A_2, \ldots, A_n \rightarrow B_m$

Why?

Splitting rule
and
Combining rule



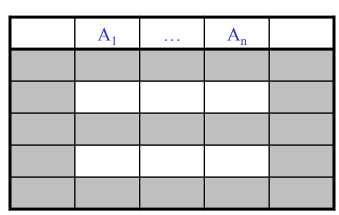
Armstrong's Rules (2/3)

$A_1, A_2, \ldots, A_n \rightarrow A_i$

Trivial Rule

where i = 1, 2, ..., n

Why?



Armstrong's Rules (3/3)

Transitive Closure Rule

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

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

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

Why?

	A_1	…	A_n		B_1	…	B_m		C_1	…	C_p	

Example (continued)

Start from the following FDs:

1. name → color
2. category → department
3. color, category → price

Infer the following FDs:

Inferred FD	Which Rule did we apply ?
4. name, category → name	
5. name, category → color	
6. name, category → category	
7. name, category → color, category	
8. name, category → price	

Example (continued)

Answers:

1. name → color
2. category → department
3. color, category → price

Inferred FD	Which Rule did we apply ?
4. name, category → name	Trivial rule
5. name, category → color	Transitivity on 4, 1
6. name, category → category	Trivial rule
7. name, category → color, category	Split/combine on 5, 6
8. name, category → price	Transitivity on 3, 7

THIS IS TOO HARD! Let's see an easier way.