Logical Database Design: Entity-Relation Models

Functional Dependencies

Schema Normalization

Alvin Cheung

Aditya Parameswaran

Reading: R & G Chapter 19

Berkeley cs186

Steps in Database Design, cont

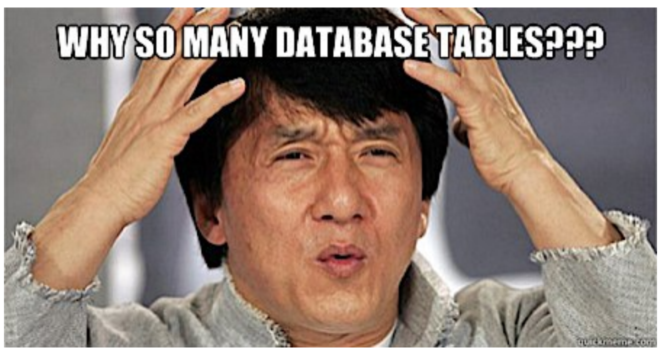
- Requirements Analysis
  - user needs; what must database do?
- Conceptual Design
  - *high level description (often done w/ER model)*
  - ORM encourages you to program here

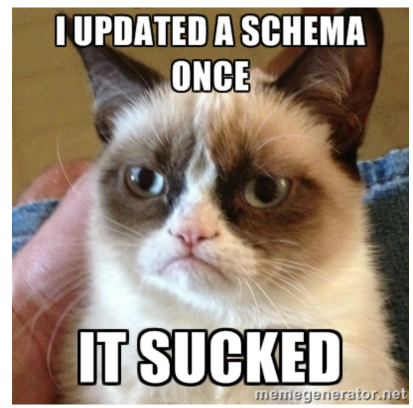
  Completed
- Logical Design
  - translate ER into DBMS data model

    Completed
  - ORMs often require you to help here too
- **Schema Refinement**

  You are here
  - **consistency, normalization**
- Physical Design - indexes, disk layout
- Security Design - who accesses what, and how

What makes good schemas?





Relational Schema Design

Name	SSN	PhoneNumber	City
Fred	123-45-6789	510-555-1234	Berkeley
Fred	123-45-6789	510-555-6543	Berkeley
Joe	987-65-4321	908-555-2121	San Jose

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

Primary key is thus (SSN, PhoneNumber)

What is the problem with this schema?

Relational Schema Design

Name	SSN	PhoneNumber	City
Fred	123-45-6789	510-555-1234	Berkeley
Fred	123-45-6789	510-555-6543	Berkeley
Joe	987-65-4321	908-555-2121	San Jose

Anomalies:

- Redundancy = repeat data
- Update anomalies = what if Fred moves to “Oakland”?
- Deletion anomalies = what if Joe deletes his phone number?

Relation Decomposition

Break the relation into two:

Name	SSN	PhoneNumber	City
Fred	123-45-6789	510-555-1234	Berkeley
Fred	123-45-6789	510-555-6543	Berkeley
Joe	987-65-4321	908-555-2121	San Jose

Name	<u>SSN</u>	City
Fred	123-45-6789	Berkeley
Joe	987-65-4321	San Jose

<u>SSN</u>	<u>PhoneNumber</u>
123-45-6789	510-555-1234
123-45-6789	510-555-6543
987-65-4321	908-555-2121

Anomalies have gone:

- No more repeated data
- Easy to move Fred to “Oakland” (how?)
- Easy to delete all Joe’s phone numbers (how?)

Relational Schema Design (or Logical Design)

How do we do this systematically?

- Start with some relational schema
- Find out its ***functional dependencies*** (FDs)
- Use FDs to ***normalize*** the relational schema

Functional Dependencies (FDs)

Berkeley cs186

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:

$A_1\ldots A_n$ **determines** $B_1..B_m$

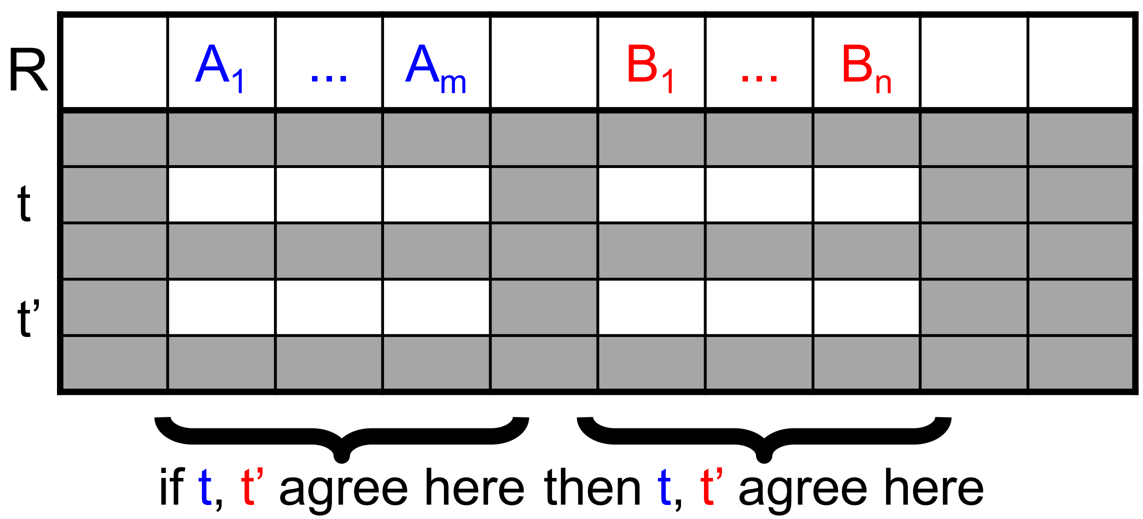
$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$

Functional Dependencies (FDs)

Definition $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ **holds** in R if:

for all t, t' ∈ R,

$(t.A_1 = t'.A_1 \wedge \ldots \wedge t.A_m = t'.A_m \rightarrow t.B_1 = t'.B_1 \wedge \ldots \wedge t.B_n = t'.B_n)$



Example

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 → Name, Phone, Position

Position → Phone

but not Phone → 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

name → color
category → department
color, category → price
department → price

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

Which FD's hold?

Buzzwords

- FD **holds** or **does not hold** on an instance
- If we can be sure that *every instance of R* will be one in which a given FD is true, then we say that **R satisfies the FD**
- If we say that R satisfies an FD, we are **stating a constraint on R**

Why bother with FDs?

Name	SSN	PhoneNumber	City
Fred	123-45-6789	510-555-1234	Berkeley
Fred	123-45-6789	510-555-6543	Berkeley
Joe	987-65-4321	908-555-2121	San Jose

Anomalies:

- Redundancy = repeat data
- Update anomalies = what if Fred moves to “Oakland”?
- Deletion anomalies = what if Joe deletes his phone number?

An Interesting Observation

If all these FDs are true:

name → color
category → department
color, category → price

Then this FD also holds:

name, category → price

If we find out from application domain that a relation satisfies some FDs, it doesn't mean that we found all the FDs that it satisfies!
There could be more FDs implied by the ones we have.

Finding New FDs: Armstrong's Axioms

- Suppose X, Y, Z are sets of attributes, then:
  - ***Reflexivity*:** If $X \supseteq Y$, then $X \rightarrow Y$
  - *Augmentation*: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
  - *Transitivity*: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

- Sound and complete inference rules for FDs!

- Some additional rules (that follow from AA):
  - *Union*: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - *Decomposition*: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
  - See if you can prove these!

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$

The **closure** is the set of attributes B, notated $\{A_1, \ldots, A_n\}^+$,
s.t. $A_1, \ldots, A_n \rightarrow B$

Example:

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

Closures:

name$^+$ = {name, color}

{name, category}$^+$ = {name, category, color, department, price}

color$^+$ = {color}

Closure Algorithm

$X=\{A_1, \ldots, A_n\}$.

Repeat until X doesn't change **do**:
if $B_1, \ldots, B_n \rightarrow C$ is a FD **and**
$B_1, \ldots, B_n$ are all in X
then add C to X.

Example:

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

$\{name, category\}^+ =$
$\{ name, category, color, department, price \}$

Hence: name, category → color, department, price

Keys

- A **superkey** is a set of attributes $A_1, \ldots, A_n$ s.t. for any other attribute B, we have $A_1, \ldots, A_n \rightarrow B$

- A **key** is a minimal superkey
    - A superkey and for which no subset is a superkey

Computing (Super)Keys

- For all sets X, compute X^+
- If X^+ = [all attributes], then X is a superkey
- Try reducing to the minimal X's to get the key

Example

Product(name, price, category, color)

name, category → price
category → color

What is the key ?

Example

Product(name, price, category, color)

name, category → price
category → color

What is the key ?

(name, category)+ = { name, category, price, color }

Hence (name, category) is a key

Key or Keys ?

We can we have more than one key!

What are the keys here ?

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

Key or Keys ?

We can we have more than one key!

What are the keys here ?

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

$AB \rightarrow C$
$BC \rightarrow A$

Eliminating Anomalies

Main idea:

- X → A is OK if X is a (super)key for the relation
- X → A is not OK otherwise
  - Need to decompose the table, but how?
  - That's where *normalization* comes in!