Relational Design Theory

Carla Teixeira Lopes

Bases de Dados
Mestrado Integrado em Engenharia Informática e Computação, FEUP

Based on Jennifer Widom and Christopher Ré slides

Agenda

Relational Design Overview

Functional Dependencies

Closures, Superkeys and Keys

Inferring Functional Dependencies

Normal Forms

Decompositions

Relational Design

Designing a database schema

- Usually many designs possible
- Some are (much) better than others!
- How do we choose?

Often use higher-level design tools, but …

- Some designers go straight to relations
- Useful to understand why tools produce certain schemas

Design theory is about how to represent your data to avoid anomalies.

Example

College application info

- SSN and name
- Colleges applying to
- High schools attended (with city)
- Hobbies

Apply (SSN, sName, cName, HS, HScity, hobby)

Example

Apply(SSN, sName, cName, HS, HScity, hobby)

123 Ann from Palo Alto High School (PAHS) and Gunn High School (GHS) also in Palo Alto plays tennis and trumpet and applied to Stanford, Berkeley, and MIT

Apply

SSN	sName	cName	HS	HScity	hobby
123	Ann	Stanford	PAHS	Palo Alto	tennis
123	Ann	Stanford	PAHS	Palo Alto	trumpet
123	Ann	Berkeley	PAHS	Palo Alto	trumpet
. .	. .	. .	. .	. .	. .

12 tuples

Design Anomalies: Redundancy

Capture information multiple times

How many times do we capture the fact that

- 123 is the social security number of Ann?
- she plays tennis?
- she applied to MIT?

Apply

SSN	sName	cName	HS	HScity	hobby
123	Ann	Stanford	PAHS	Palo Alto	tennis
123	Ann	Stanford	PAHS	Palo Alto	trumpet
123	Ann	Berkeley	PAHS	Palo Alto	trumpet
. .	. .	. .	. .	. .	. .

Design Anomalies: Update Anomaly

Direct effect of redundancy

Can update facts in some places but not all or differently in different places

~~trumpet~~ -> piano

Apply

SSN	sName	cName	HS	HScity	hobby
123	Ann	Stanford	PAHS	Palo Alto	tennis
123	Ann	Stanford	PAHS	Palo Alto	piano
123	Ann	Berkeley	PAHS	Palo Alto	trumpet
. .	. .	. .	. .	. .	. .

Inconsistent database

Design Anomalies: Deletion Anomaly

Inadvertently deletion

Example

- 1 new tuple about John
- Someone decides surfing is an unacceptable hobby and delete the tuples about surfing
- This will completely delete students with surfing as their only hobby

Apply

SSN	sName	cName	HS	HScity	hobby
123	Ann	Stanford	PAHS	Palo Alto	trumpet
123	Ann	Berkeley	PAHS	Palo Alto	trumpet
.	.	.	.	.	.
234	John	MIT	PAHS	Palo Alto	surfing

Example: New Design

College application info

- SSN and name / Colleges applying to / High schools attended (with city) / Hobbies

What about this design?

- Student (SSN, sName)
- Apply (SSN, cName)
- HighSchool (SSN, HS)
- Located (HS, HScity)
- Hobbies (SSN, hobby)

No redundancy
No update or deletion anomalies
Reconstruction of original data

We'll understand why this design is better and learn how to find this decomposition

Example: Modifications to the New Design

What if the high school name alone is not a key?

Student (SSN, sName)

Apply (SSN, cName)

HighSchool (SSN, HS)

Located (HS, HScity)

Hobbies (SSN, hobby)

→

Student (SSN, sName)

Apply (SSN, cName)

HighSchool (SSN, HS, HScity)

~~Located (HS, HScity)~~

Hobbies (SSN, hobby)

Example: Modifications to the New Design

What if students don't want all of their hobbies revealed to all of the colleges?

Student (SSN, sName)

Apply (SSN, cName)

HighSchool (SSN, HS, HScity)

~~Located (HS, HScity)~~

Hobbies (SSN, hobby)

→

Student (SSN, sName)

Apply (SSN, cName, hobby)

HighSchool (SSN, HS, HScity)

~~Located (HS, HScity)~~

~~Hobbies (SSN, hobby)~~

The best design also depends in what the data is representing in the real world

Design by decomposition

Start with “mega” relations containing everything

Decompose into smaller, better relations with same information

Decomposition can be done automatically

- “Mega” relations + properties of the data → Functional dependencies
- System decomposes based on properties
- Final set of relations satisfies normal form → No anomalies, no lost information

Agenda

~~Relational Design Overview~~

Functional Dependencies

Closures, Superkeys and Keys

Inferring Functional Dependencies

Normal Forms

Decompositions

Functional Dependencies (FD)

Generalization of the notion of keys

Relational design by decomposition

- To design a better schema, one which minimizes the possibility of anomalies

Data storage

- Compression schemes based on functional dependencies can be used

Reasoning about queries

- Optimization of queries

Example

Student (SSN, sName, address, HScode, HSname,
HScity, GPA, priority)

Suppose priority is determined by GPA

- $GPA > 3.8 \rightarrow priority = 1$
- $3.3 < GPA <= 3.8 \rightarrow priority = 2$
- $GPA <= 3.3 \rightarrow priority = 3$

Two tuples with same GPA have same priority

Example

Student (SSN, sName, address, HScode, HSname,
HScity, GPA, priority)

Two tuples with same GPA have same priority

$$\forall\, t, u \in Student: t.GPA = u.GPA \Rightarrow t.priority = u.priority$$

GPA -> priority

Definition

A and B are attributes of a relation R

A -> B

A functionally determines B

$$\forall\ t, u \in R\colon t.A = u.A \Rightarrow t.B = u.B$$

Definition

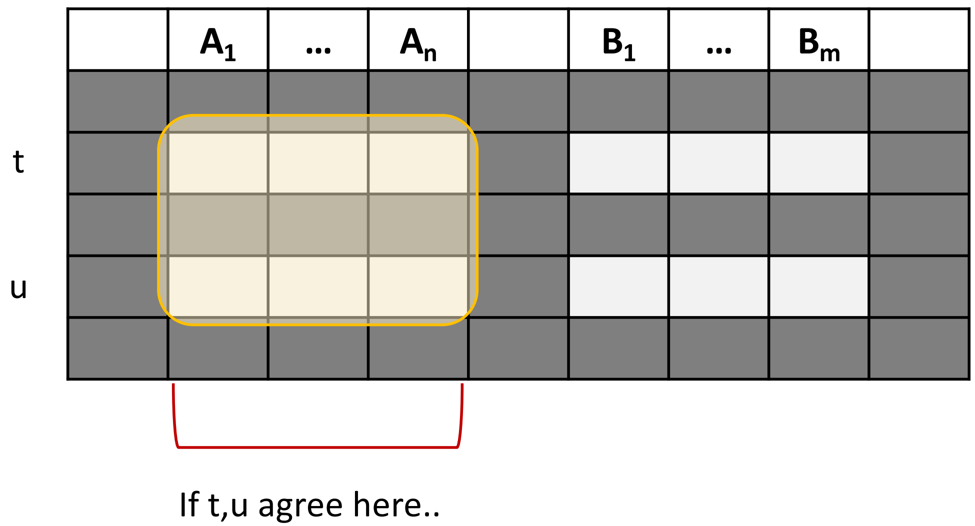
$A_1, A_2, \ldots, A_n$ and $B_1, B_2, \ldots, B_m$ are attributes of a relation R

$$\underbrace{A_1, A_2, \ldots, A_n}_{\bar{A}} \text{ -> } \underbrace{B_1, B_2, \ldots, B_m}_{\bar{B}}$$

$$\forall\ t, u \in R: t[A1, \ldots, An] = u[A1, \ldots, An] \Rightarrow t.[B1, \ldots, Bm] = u.[B1, \ldots, Bm]$$

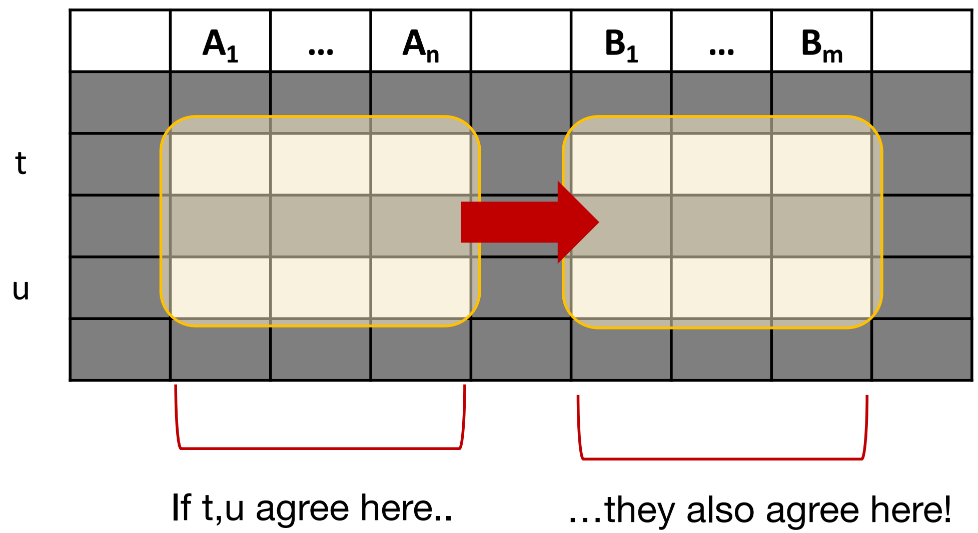
A Picture of FDs

$$\forall\, t, u \in R: t[A1, \ldots, An] = u[A1, \ldots, An] \Rightarrow t.[B1, \ldots, Bm] = u.[B1, \ldots, Bm]$$



A Picture of FDs

$$\forall\, t, u \in R: t[A1, \ldots, An] = u[A1, \ldots, An] \Rightarrow t.[B1, \ldots, Bm] = u.[B1, \ldots, Bm]$$



Identifying FDs

Based on knowledge of real world

All instances of relation must adhere

You can check if an FD is violated by examining a single instance

However, you cannot prove that an FD is part of the schema by examining a single instance

- This would require checking every valid instance

Example 1

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

What functional dependencies do you identify?

Example 1

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

{Position} → {Phone}

Example 2

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 3

Student (SSN, sName, address, HScode, HSname, HScity, GPA, priority)

SSN -> sName

SSN -> address

HScode -> HSname, HScity

HSname, HScity -> HScode

SSN -> GPA

GPA -> priority

SSN -> priority

Assuming the student doesn't move

No two high schools with the same name in the same city

Example 4

Apply (SSN, cName, state, date, major)

cName -> date → Assuming every college has a single date to receive applications

SSN, cName -> major → Assuming students are only allowed to apply to a single major at each college

Depends on the real world constraints

Keys

Relation with no duplicates: $\mathrm{R}(\bar{A}, \bar{B})$

If $\bar{A}$ -> all attributes then $\bar{A}$ is a key

R

$\bar{A}$	$\bar{B}$
$\bar{a}$	$\overline{b_1}$
$\bar{a}$	$\overline{b_2}$
.	.

$\bar{A} \rightarrow all$

R

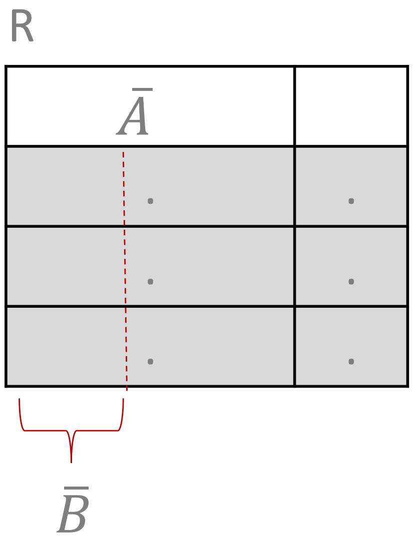
$\bar{A}$	$\bar{B}$
$\bar{a}$	$\bar{b}$
$\bar{a}$	$\bar{b}$
.	.

We cannot have two separate tuples with the same $\bar{a}$ values

$\bar{A}$ is a key

Trivial Functional Dependency

$\bar{A} \rightarrow \bar{B}$ is a trivial dependency if $\bar{B} \subseteq \bar{A}$

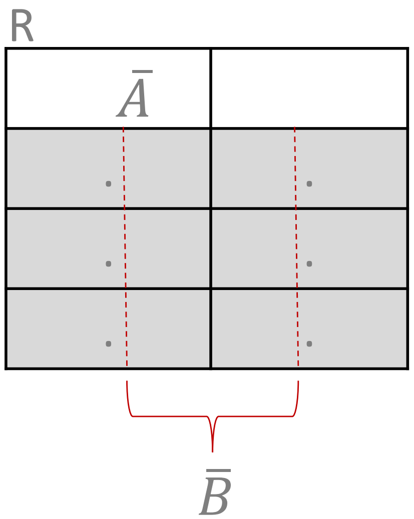


It's obvious that, if $\bar{A}$ has the same values, then $\bar{B}$ will have the same values

Nontrivial Functional Dependency

A functional dependency that's not a trivial one

$\bar{A} \rightarrow \bar{B}$ is a nontrivial dependency if $\bar{B} \not\subseteq \bar{A}$



Now the FD is saying something

Completely Nontrivial Functional Dependency

A functional dependency that's not a trivial one

$\bar{A} \rightarrow \bar{B}$ is a completely nontrivial dependency if $\bar{A} \cap \bar{B} = \emptyset$

R

$\bar{A}$	$\bar{B}$	
.		
.		
.		

These are the ones we're most interested in

Finding Functional Dependencies

Given a set of FDs, $F = \{f_1, \ldots f_n\}$, does an FD g hold?

How do we decide?

Using rules

- Splitting/Combining, Trivial and Transitivity

Splitting Rule

We can split the right side of the FD

$$\bar{A} \rightarrow B_1, B_2, \dots, B_n \Rightarrow \bar{A} \rightarrow B_1, \bar{A} \rightarrow B_2, \dots, \bar{A} \rightarrow B_n$$

Can we also split left-hand-side?

$A_1, A_2, \dots, A_n \rightarrow \bar{B} \Rightarrow A_1 \rightarrow \bar{B}, A_2 \rightarrow \bar{B}, \dots, A_2 \rightarrow \bar{B}$?

No, for example

HSname, HScity -> HScode

~~HSname -> HScode~~

~~HScity -> HScode~~

Combining Rule

Inverse of the splitting rule

$$\bar{A} \to B_1, \bar{A} \to B_2, \dots, \bar{A} \to B_n \Rightarrow \bar{A} \to B_1, B_2, \dots, B_n$$

Trivial-dependency rules

Reminder: $\bar{A} \to \bar{B}$ is a trivial dependency if $\bar{B} \subseteq \bar{A}$

Every left hand side determines itself or any subset of itself

$\bar{A} \to \bar{B} \;\Rightarrow \bar{A} \to \bar{A} \cup \bar{B}$

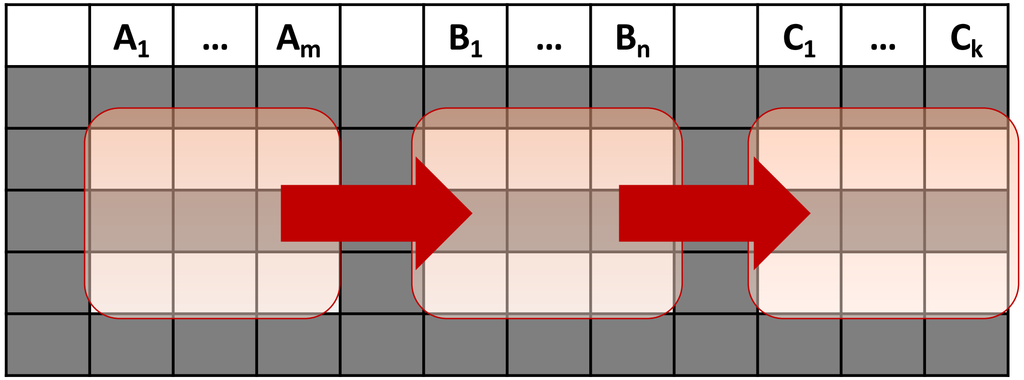
We can add to the right side of every FD what's already on the left hand side

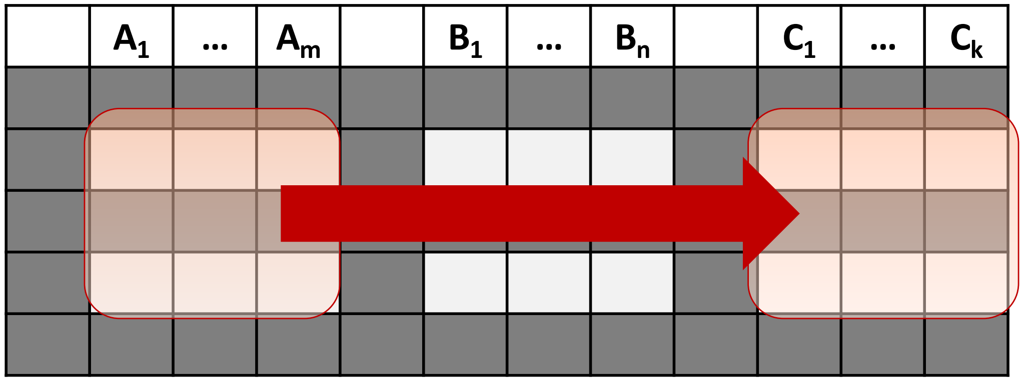
$\bar{A} \to \bar{B} \;\Rightarrow \bar{A} \to \bar{A} \cap \bar{B}$

Also implied by the splitting rule

Transitive Rule

$$\bar{A} \rightarrow \bar{B} \ \wedge \ \bar{B} \rightarrow \bar{C} \Rightarrow \bar{A} \rightarrow \bar{C}$$





Transitive Rule

$$\bar{A} \rightarrow \bar{B} \ \wedge \bar{B} \rightarrow \bar{C} \Rightarrow \bar{A} \rightarrow \bar{C}$$

$\bar{A} \rightarrow \bar{B}$

$\bar{A}$	$\bar{B}$	$\bar{C}$	$\bar{D}$
$\bar{a}$	$\bar{b}$		
$\bar{a}$	$\bar{b}$		

$\bar{B} \rightarrow \bar{C}$

$\bar{A}$	$\bar{B}$	$\bar{C}$	$\bar{D}$
$\bar{a}$	$\bar{b}$	$\bar{c}$	
$\bar{a}$	$\bar{b}$	$\bar{c}$	

Shows that $\bar{A} \rightarrow \bar{C}$ holds

Finding Functional Dependencies

Products

Name	Color	Category	Dep	Price
Gizmo	Green	Gadget	Toys	49
Widget	Black	Gadget	Toys	59
Gizmo	Green	Whatsit	Garden	99

Provided FDs

1. {Name} -> {Color}
2. {Category} -> {Department}
3. {Color, Category} -> {Price}

Inferred FD	Rule used
4. {Name, Category} -> {Name}	?
5. {Name, Category} -> {Color}	?
6. {Name, Category} -> {Category}	?
7. {Name, Category} -> {Color, Category}	?
8. {Name, Category} -> {Price}	?

Finding Functional Dependencies

Products

Name	Color	Category	Dep	Price
Gizmo	Green	Gadget	Toys	49
Widget	Black	Gadget	Toys	59
Gizmo	Green	Whatsit	Garden	99

Provided FDs

1. {Name} -> {Color}
2. {Category} -> {Department}
3. {Color, Category} -> {Price}

Inferred FD	Rule used
4. {Name, Category} -> {Name}	Trivial
5. {Name, Category} -> {Color}	Transitive (4, 1)
6. {Name, Category} -> {Category}	Trivial
7. {Name, Category} -> {Color, Category}	Combining rule (5, 6)
8. {Name, Category} -> {Price}	Transitive (7, 3)

Kahoot time!

Any doubts?

Kahoot time!

This relation has redundancy because ...

- the product name is determined by its id.
- the order date may be repeated.
- it has no client id.
- the name of the product with id=5 is stored more than once.

```
Order (OrderID, OrderDate, ClientName, ProductID, ProductName)
```

Kahoot time!

Identify a deletion anomaly.

- Removing old orders, you may lose clients.
- Removing products with id<100, you may lose their names.
- Updating old orders, you may lose clients.
- Removing the attribute OrderDate, you may lose clients.

```
Order (OrderID, OrderDate, ClientName, ProductID, ProductName)
```

Kahoot time!

The set of relations created by decomposition for preventing anomalies is called:

- normal forms.
- mega-relations.
- functional dependencies.
- None of the above is correct.

Kahoot time!

Identify the correct statement.

- OrderID -> OrderDate, ClientName, ProductID, ProductName
- OrderID -> ProductID, ProductName
- OrderID -> OrderDate
- OrderDate -> OrderID

```
Order (OrderID, OrderDate, ClientName, ProductID, ProductName)
```

Readings

Jeffrey Ullman, Jennifer Widom, A first course in Database Systems 3rd Edition

- Section 3.1 – Functional Dependencies
- Section 3.2 – Rules About Functional Dependencies
- Section 3.3 – Design of Relational Database Schemas
- Section 3.4 – Decomposition: The Good, Bad, and Ugly
- Section 3.5 – Third Normal Form