Database Design: Normalization

Agenda

1. Database Design

2. Normal forms & functional dependencies

3. Finding functional dependencies

4. Closures, superkeys & keys

5. Relation Decomposition

FINDING FUNCTIONAL DEPENDENCIES

What you will learn about in this section

1. “Good” vs. “Bad” FDs: Intuition
2. Finding FDs
3. Closures

“Good” vs. “Bad” FDs

We can start to develop a notion of **good** vs. **bad** FDs:

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

<u>Intuitively:</u>

EmpID -> Name, Phone,Position is *“good FD”*

Minimal redundancy, less possibility of anomalies

“Good” vs. “Bad” FDs

We can start to develop a notion of **good** vs. **bad** FDs:

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

<u>Intuitively:</u>

EmpID -> Name, Phone, Position is *“good FD”*

But Position -> Phone *is a “bad FD”*

Redundancy!
Possibility of data anomalies

“Good” vs. “Bad” FDs

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

Returning to our original example… can you see how the “bad FD” {Course} -> {Room} could lead to an:

- Update Anomaly
- Insert Anomaly
- Delete Anomaly
- …

Given a set of FDs (from user) our goal is to:

1. **Find all FDs, and**
2. **Eliminate the “Bad Ones".**

Finding Functional Dependencies

- There can be a very **large number** of FDs…
  - How to find them all efficiently?
- We can't necessarily show that any FD will hold **on all instances…**
  - How to do this?

We will start with this problem:
Given a set of FDs, F, what other FDs ***must*** hold?

Finding Functional Dependencies

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

Inference problem: How do we decide?

Finding Functional Dependencies

Example:

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}

Given the provided FDs, we can see that {Name, Category} → {Price} must also hold on **any instance**…

Which / how many other FDs do?!?

Finding Functional Dependencies

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

Inference problem: How do we decide?

Axioms:

Reflexivity: if $Y \subseteq X$, then $X \rightarrow Y$
Augmentation: if $X \rightarrow Y$, then $WX \rightarrow WY$
Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Derived Rules:

Union: if $X \rightarrow Y$ and $X \rightarrow Z$, the $X \rightarrow YZ$
Decomposition: if $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
Pseudo transitivity: if $X \rightarrow Y$ and $WY \rightarrow Z$, then $XW \rightarrow Z$

Finding Functional Dependencies

Example:

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}

Which / how many other FDs hold?

Finding Functional Dependencies

Example:

Provided FDs:

1. {Name} → {Color}
2. {Category} → {Dept.}
3. {Color, Category} → {Price}

Inferred FDs:

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}	?

Which / how many other FDs hold?

Finding Functional Dependencies

Provided FDs:

1. {Name} → {Color}
2. {Category} →{Dept.}
3. {Color, Category} →{Price}

<u>Example:</u>

Inferred FDs:

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}	Split/combine (5 + 6)
8. {Name, Category} -> {Price}	Transitive (7 -> 3)

Can we find an algorithmic way to do this?

Yes. But we need to learn about closures before that!

Closures

Closure of a set of Attributes

Given a set of attributes $\mathbf{A_1, \ldots, A_n}$ and a set of FDs **F:**
Then the **closure**, $\mathbf{\{A_1, \ldots, A_n\}^+}$ is the set of attributes **B** s.t. $\{\mathbf{A_1, \ldots, A_n}\} \rightarrow \mathbf{B}$

Example: F =

```
{name} → {color}
{category} → {department}
{color, category} → {price}
```

Example Closures:

```
{name}+ = {name, color}
{name, category}+ =
{name, category, color, dept, price}
{color}+ = {color}
```

Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$ and set of FDs F.

Repeat until X doesn't change;

do:

if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$ **then**

add C to X.

Return X as X^+

Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change;
do:
  if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$:
    then add C to X.
Return X as X^+

F =

```
{name} → {color}

{category} → {dept}

{color, category} → {price}
```

```
{name, category}+ =
{name, category}
```

Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change;
do:
  if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$:
    then add C to X.
Return X as X^+

F =

`{name} → {color}`

`{category} → {dept}`

`{color, category} → {price}`

`{name, category}+ = {name, category}`

`{name, category}+ = {name, category, color}`

Closure Algorithm

Start with X = $\{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change;
do:
  if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$:
    then add C to X.
Return X as X^+

F =

```
{name} → {color}

{category} → {dept}

{color, category} →
{price}
```

```
{name, category}+ =
{name, category}
```

```
{name, category}+ =
{name, category, color}
```

```
{name, category}+ =
{name, category, color, dept}
```

Closure Algorithm

Start with X = $\{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change;
do:
  if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$:
    then add C to X.
Return X as X^+

F =

`{name} → {color}`

`{category} → {dept}`

`{color, category} → {price}`

`{name, category}+ = {name, category}`

`{name, category}+ = {name, category, color}`

`{name, category}+ = {name, category, color, dept}`

`{name, category}+ = {name, category, color, dept, price}`

EXAMPLE

R(A,B,C,D,E,F)

$\{A,B\} \rightarrow \{C\}$
$\{A,D\} \rightarrow \{E\}$
$\{B\} \rightarrow \{D\}$
$\{A,F\} \rightarrow \{B\}$

Compute $\{A, B\}^{+} = \{A, B,$ $\}$

Compute $\{A, F\}^{+} = \{A, F,$ $\}$

EXAMPLE

R(A,B,C,D,E,F)

{A,B} → {C}
{A,D} → {E}
{B} → {D}
{A,F} → {B}

Compute $\{A, B\}^+ = \{A, B, C, D \qquad \}$

Compute $\{A, F\}^+ = \{A, F, B \qquad \}$

EXAMPLE

R(A,B,C,D,E,F)

$\{A,B\} \rightarrow \{C\}$
$\{A,D\} \rightarrow \{E\}$
$\{B\} \rightarrow \{D\}$
$\{A,F\} \rightarrow \{B\}$

Compute $\{A, B\}^+ = \{A, B, C, D, E\}$

Compute $\{A, F\}^+ = \{A, B, C, D, E, F\}$

3. CLOSURES, SUPERKEYS & KEYS

What you will learn about in this section

1. Closures
2. Superkeys & Keys

Why Do We Need the Closure?

- With closure we can find all FD's easily

- To check if $X \to A$

  1. Compute X^+

  2. Check if $A \in X^+$

Note here that **X** is a *set* of attributes, but **A** is a *single* attribute. Why does considering FDs of this form suffice?

Using Closure to Infer ALL FDs

Example:
Given F =

$\{A,B\} \rightarrow C$
$\{A,D\} \rightarrow B$
$\{B\} \rightarrow D$

Step 1: Compute X^+, for every set of attributes X:

$\{A\}^+ = \{A\}$
$\{B\}^+ = \{B,D\}$
$\{C\}^+ = \{C\}$
$\{D\}^+ = \{D\}$
$\{A,B\}^+ = \{A,B,C,D\}$
$\{A,C\}^+ = \{A,C\}$
$\{A,D\}^+ = \{A,B,C,D\}$
$\{A,B,C\}^+ = \{A,B,D\}^+ = \{A,C,D\}^+ = \{A,B,C,D\}$
$\{B,C,D\}^+ = \{B,C,D\}$
$\{A,B,C,D\}^+ = \{A,B,C,D\}$

No need to compute these- why?

We did not include {B,C}, {B,D}, {C,D}, {B,C,D} to save some space.

Using Closure to Infer ALL FDs

Example:

Given F =

$\{A,B\} \rightarrow C$
$\{A,D\} \rightarrow B$
$\{B\} \rightarrow D$

Step 1: Compute X^+, for every set of attributes X:

$\{A\}^+ = \{A\}$, $\{B\}^+ = \{B,D\}$, $\{C\}^+ = \{C\}$, $\{D\}^+ = \{D\}$, $\{A,B\}^+ = \{A,B,C,D\}$, $\{A,C\}^+ = \{A,C\}$, $\{A,D\}^+ = \{A,B,C,D\}$, $\{A,B,C\}^+ = \{A,B,D\}^+ = \{A,C,D\}^+ = \{A,B,C,D\}$, $\{B,C,D\}^+ = \{B,C,D\}$, $\{A,B,C,D\}^+ = \{A,B,C,D\}$

Step 2: Enumerate all FDs $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

$\{A,B\} \rightarrow \{C,D\}$, $\{A,D\} \rightarrow \{B,C\}$,
$\{A,B,C\} \rightarrow \{D\}$, $\{A,B,D\} \rightarrow \{C\}$,
$\{A,C,D\} \rightarrow \{B\}$

Using Closure to Infer ALL FDs

Example:
Given F =

$\{A,B\} \rightarrow C$
$\{A,D\} \rightarrow B$
$\{B\} \rightarrow D$

Step 1: Compute X^+, for every set of attributes X:

$\{A\}^+ = \{A\}$, $\{B\}^+ = \{B,D\}$, $\{C\}^+ = \{C\}$, $\{D\}^+ = \{D\}$, $\{A,B\}^+ = \{A,B,C,D\}$, $\{A,C\}^+ = \{A,C\}$, $\{A,D\}^+ = \{A,B,C,D\}$, $\{A,B,C\}^+ = \{A,B,D\}^+ = \{A,C,D\}^+ = \{A,B,C,D\}$, $\{B,C,D\}^+ = \{B,C,D\}$, $\{A,B,C,D\}^+ = \{A,B,C,D\}$

Step 2: Enumerate all FDs $X \rightarrow Y$, s.t. $\boxed{Y \subseteq X^+}$ and $X \cap Y = \varnothing$:

$\{A,B\} \rightarrow \{C,D\}$, $\{A,D\} \rightarrow \{B,C\}$, $\{A,B,C\} \rightarrow \{D\}$, $\{A,B,D\} \rightarrow \{C\}$, $\{A,C,D\} \rightarrow \{B\}$

"Y is in the closure of X"

Using Closure to Infer ALL FDs

<u>Example:</u>

Given F =

$\{A,B\} \rightarrow C$
$\{A,D\} \rightarrow B$
$\{B\} \rightarrow D$

Step 1: Compute X^+, for every set of attributes X:

$\{A\}^+ = \{A\}$, $\{B\}^+ = \{B,D\}$, $\{C\}^+ = \{C\}$, $\{D\}^+ = \{D\}$, $\{A,B\}^+ = \{A,B,C,D\}$, $\{A,C\}^+ = \{A,C\}$, $\{A,D\}^+ = \{A,B,C,D\}$, $\{A,B,C\}^+ = \{A,B,D\}^+ = \{A,C,D\}^+ = \{A,B,C,D\}$, $\{B,C,D\}^+ = \{B,C,D\}$, $\{A,B,C,D\}^+ = \{A,B,C,D\}$

Step 2: Enumerate all FDs $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

$\{A,B\} \rightarrow \{C,D\}$, $\{A,D\} \rightarrow \{B,C\}$,
$\{A,B,C\} \rightarrow \{D\}$, $\{A,B,D\} \rightarrow \{C\}$,
$\{A,C,D\} \rightarrow \{B\}$

The FD $X \rightarrow Y$ is non-trivial

Superkeys and Keys

Keys and Superkeys

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

I.e. all attributes are *functionally determined* by a superkey

A **key** is a *minimal* superkey

Meaning that no subset of a key is also a superkey

Finding Keys and Superkeys

- For each set of attributes X

1. Compute X^+
2. If X^+ = set of all attributes then X is a **superkey**
3. If X is minimal, then it is a **key**

Do we need to check all sets of attributes?

Example of Finding Keys

```
Product(name, price, category, color)
```

```
{name, category} → price
{category} → color
```

What is a key?

Example of Keys

```
Product(name, price, category, color)
```

```
{name, category} → price
{category} → color
```

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

= the set of all attributes

$\Longrightarrow$ this is a **superkey**

$\Longrightarrow$ this is a **key**, since neither name nor category alone is a superkey

Decompositions

Decompositions

Decomposition of a relation is done when a relation in relational model is not in appropriate normal form.

Relation R is decomposed into two or more relations if decomposition is **lossless join** as well as **dependency preserving**.

Decompositions

If R(A, B, C) satisfies $A \rightarrow B$

We can project it on A, B and A,C ***without losing information***

Lossless decomposition vs. **Lossy** decomposition

If we decompose a relation R(A, B, C) into relations

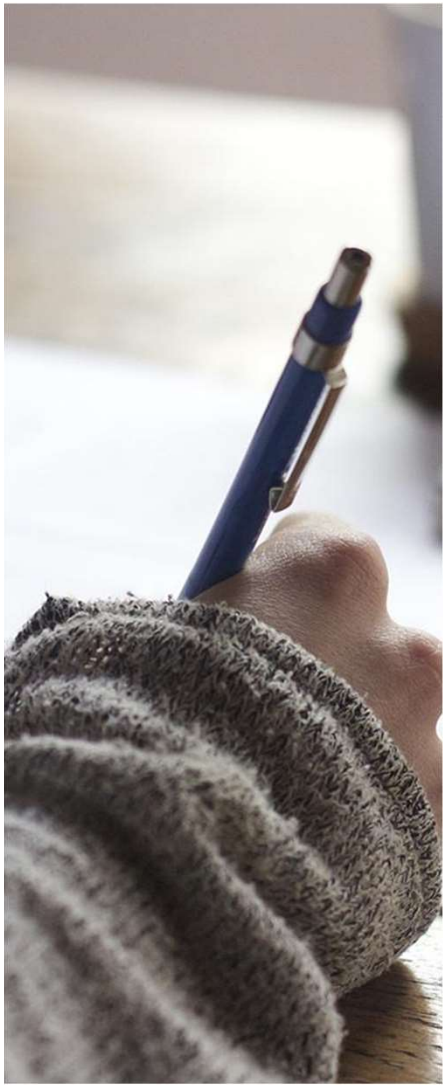
$R1 = \pi_{AB}(R)$ and $R2 = \pi_{AC}(R)$

$\pi_{AB}(R)$ is the projection of R on AB

$\bowtie$ is the natural join operator

Decomposition is **lossy** if $R \subset R1 \bowtie R2$

Decomposition is **lossless** if $R = R1 \bowtie R2$



Decompositions

$$\begin{array}{c} R(A_1,\dots,A_n,B_1,\dots,B_m,C_1,\dots,C_p) \\ \swarrow \qquad\qquad \searrow \\ R_1(A_1,\dots,A_n,B_1,\dots,B_m) \qquad R_2(A_1,\dots,A_n,C_1,\dots,C_p) \end{array}$$

R_1 = the *projection* of R on $A_1, \dots, A_n, B_1, \dots, B_m$

R_2 = the *projection* of R on $A_1, \dots, A_n, C_1, \dots, C_p$

Properties of Decomposition

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

Name	Price
Gizmo	19.99
OneClick	24.99
Gizmo	19.99

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

We need a decomposition to be "correct"

I.e. it is a **Lossless decomposition**

Lossy Decomposition

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

Need to avoid "bad" decompositions

What's wrong here?

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

Lossy Decomposition

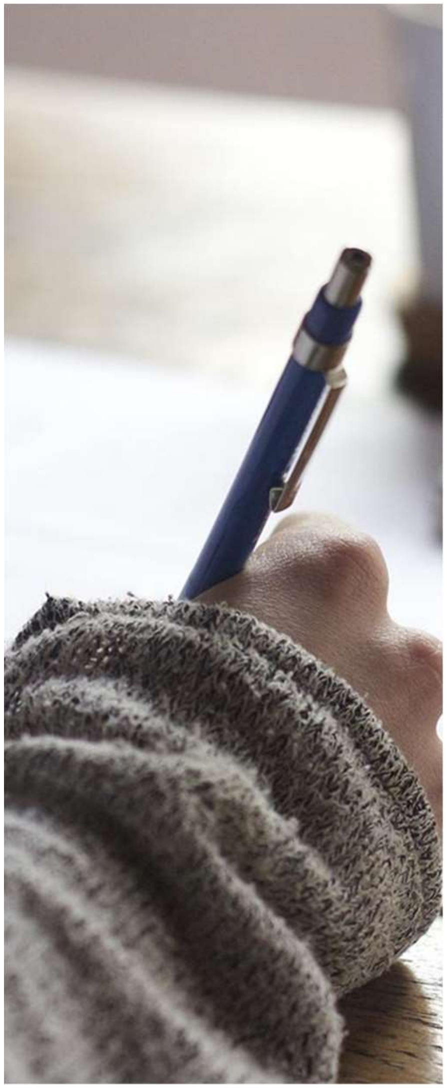
Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

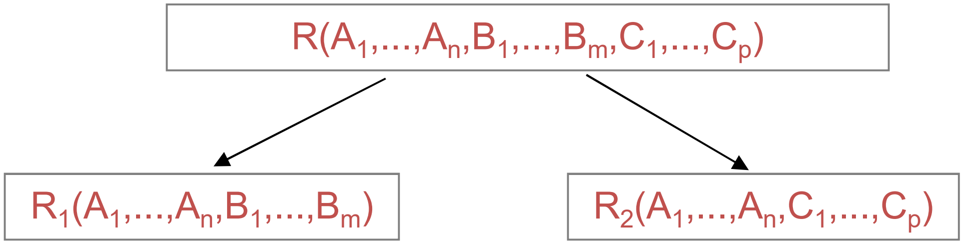
$\bowtie$

Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

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



Lossless Decompositions



A decomposition R to (R1, R2) is **lossless** if R = R1 $\bowtie$ R2

To check for lossless join decomposition using FD set, following conditions must hold:

1- Union of Attributes of R1 and R2 must be equal to attribute of R. Each attribute of R must be either in R1 or in R2.

$$Att(R1) \cup Att(R2) = Att(R)$$

2- Intersection of Attributes of R1 and R2 must not be NULL.

$$Att(R1) \cap Att(R2) \neq \Phi$$

3- Common attribute must be a key for at least one relation (R1 or R2).

$$Att(R1) \cap Att(R2) \rightarrow Att(R1)$$

or

$$Att(R1) \cap Att(R2) \rightarrow Att(R2)$$

Example

A relation R (A, B, C, D) with FD set { A -> BC} is decomposed into R1(ABC) and R2(AD)

Is lossless join decomposition?

First condition holds **true** as Att(R1) U Att(R2) = (ABC) U (AD) = (ABCD) = Att(R).

Second condition holds **true** as Att(R1) ∩ Att(R2) = (ABC) ∩ (AD) ≠ Φ

Third condition holds **true** as Att(R1) ∩ Att(R2) = A is a key of R1(ABC) because A->BC is given.

Dependency Preserving Decomposition

If we decompose a relation R into relations R1 and R2, All dependencies of R either must be a part of R1 or R2 or must be derivable from combination of FD's of R1 and R2.

For Example, A relation R (A, B, C, D) with FD set { A -> BC} is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A -> BC is a part of R1(ABC).

Question

Consider a schema R(A,B,C,D) and functional dependencies A->B and C->D. Then the decomposition of R into R1(AB) and R2(CD) is

A. dependency preserving and lossless join

B. lossless join but not dependency preserving

C. dependency preserving but not lossless join

D. not dependency preserving and not lossless join

Answer

For **lossless join** decomposition, these three conditions must hold true:

Att(R1) U Att(R2) = ABCD = Att(R)

Att(R1) ∩ Att(R2) = Φ, which violates the condition of lossless join decomposition. Hence the decomposition is not lossless.

For **dependency preserving** decomposition,

A -> B can be ensured in R1(AB) and C -> D can be ensured in R2(CD). Hence it is dependency preserving decomposition.

So, the correct option is C.

Acknowledgement

Some of these slides are taken from cs145 course offered by Stanford University.