

# FUNCTIONAL DEPENDENCIES

---

*CS 564- Fall 2021*

---

*ACKs: Dan Suciu, Jignesh Patel, AnHai Doan*

---

# WHAT IS THIS LECTURE ABOUT?

---

Database Design Theory:

- Functional Dependencies
- Armstrong's rules
- The Closure Algorithm
- Keys and Superkeys

# HOW TO BUILD A DB APPLICATION

- Pick an application
- Figure out what to model (**ER model**)
  - Output: **ER diagram**
- Transform the ER diagram to a **relational schema**

• Refine the relational schema (**normalization**)

- Now ready to implement the schema and load the data!

---

# DB DESIGN THEORY

---

- Helps us identify the “bad” schemas and improve them
  1. express constraints on the data: **functional dependencies (FDs)**
  2. use the FDs to decompose the relations
- The process, called **normalization**, obtains a schema in a “normal form” that guarantees certain properties
  - examples of normal forms: **BCNF, 3NF, ...**

# MOTIVATING EXAMPLE

| SSN       | name  | age | phoneNumber  |
|-----------|-------|-----|--------------|
| 934729837 | Paris | 24  | 608-374-8422 |
| 934729837 | Paris | 24  | 603-534-8399 |
| 123123645 | John  | 30  | 608-321-1163 |
| 384475687 | Arun  | 20  | 206-473-8221 |

- What is the primary key?
  - (SSN, PhoneNumber)
- What is the problem with this schema?
  - Age and name are stored redundantly

# MOTIVATING EXAMPLE

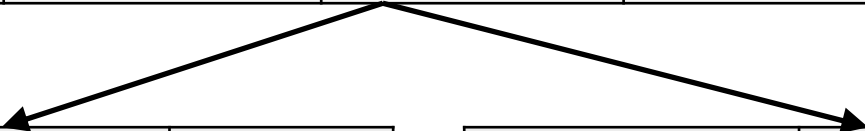
| SSN       | name  | age | phoneNumber  |
|-----------|-------|-----|--------------|
| 934729837 | Paris | 24  | 608-374-8422 |
| 934729837 | Paris | 24  | 603-534-8399 |
| 123123645 | John  | 30  | 608-321-1163 |
| 384475687 | Arun  | 20  | 206-473-8221 |

## Problems:

- redundant storage
- **update**: change the age of Paris?
- **insert**: what if a person has no phone number?
- **delete**: what if Arun deletes his phone number?

# SOLUTION: DECOMPOSITION

| SSN       | name  | age | phoneNumber  |
|-----------|-------|-----|--------------|
| 934729837 | Paris | 24  | 608-374-8422 |
| 934729837 | Paris | 24  | 603-534-8399 |
| 123123645 | John  | 30  | 608-321-1163 |
| 384475687 | Arun  | 20  | 206-473-8221 |



| SSN       | name  | age |
|-----------|-------|-----|
| 934729837 | Paris | 24  |
| 123123645 | John  | 30  |
| 384475687 | Arun  | 20  |

| SSN       | phoneNumber  |
|-----------|--------------|
| 934729837 | 608-374-8422 |
| 934729837 | 603-534-8399 |
| 123123645 | 608-321-1163 |
| 384475687 | 206-473-8221 |

---

# FUNCTIONAL DEPENDENCIES

---



# FD: DEFINITION

- **Functional dependencies** (FDs) are a form of **constraint**
- they generalize the concept of keys

If two tuples agree on the attributes

$$A = A_1, A_2, \dots, A_n$$

then they must agree on the attributes

$$B = B_1, B_2, \dots, B_m$$

Formally:

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

We then say that  $A$  **functionally determines**  $B$

# FD: EXAMPLE 1

| SSN       | name  | age | phoneNumber  |
|-----------|-------|-----|--------------|
| 934729837 | Paris | 24  | 608-374-8422 |
| 934729837 | Paris | 24  | 603-534-8399 |
| 123123645 | John  | 30  | 608-321-1163 |
| 384475687 | Arun  | 20  | 206-473-8221 |

- $SSN \rightarrow name, age$
- $SSN, age \rightarrow name$

# FD: EXAMPLE 2

| studentID | semester | courseNo | section | instructor |
|-----------|----------|----------|---------|------------|
| 124434    | 4        | CS 564   | 1       | Paris      |
| 546364    | 4        | CS 564   | 2       | Arun       |
| 999492    | 6        | CS 764   | 1       | Anhai      |
| 183349    | 6        | CS 784   | 1       | Jeff       |

- *courseNo, section*  $\rightarrow$  *instructor*
- *studentID*  $\rightarrow$  *semester*

# SPLITTING AN FD

---

- Consider the FD:  $A, B \rightarrow C, D$
- The attributes on the right are **independently** determined by  $A, B$  so we can split the FD into:
  - $A, B \rightarrow C$  and  $A, B \rightarrow D$
- We can not do the same with attributes on the left!
  - writing  $A \rightarrow C, D$  and  $B \rightarrow C, D$  does **not** express the same constraint!

# TRIVIAL FDS

---

- Not all FDs are informative:
  - $A \rightarrow A$  holds for any relation
  - $A, B, C \rightarrow C$  also holds for any relation
- An FD  $X \rightarrow A$  is called **trivial** if the attribute  $A$  belongs in the attribute set  $X$ 
  - a trivial FD always holds!

---

# HOW TO IDENTIFY FDS

---

- An FD is domain knowledge:
  - an inherent property of the application & data
  - not something we can infer from a set of tuples
- Given a table with a set of tuples
  - we can confirm that a FD **seems** to be valid
  - to infer that a FD is **definitely** invalid
  - we can **never** prove that a FD is valid

# EXAMPLE 3

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

**Q1:** Is  $name \rightarrow department$  an FD?

– not possible!

**Q2:** Is  $name, category \rightarrow department$  an FD ?

– we don't know!

# WHY FDS?

---

1. keys are special cases of FDs
2. more integrity constraints for the application
3. having FDs will help us detect that a schema has redundancies and tell us how to normalize it



# MORE ON FDS

---

- If the following FDs hold:

- $A \longrightarrow B$

- $B \longrightarrow C$

then the following FD is **also** true:

- $A \longrightarrow C$

- We can find more FDs like that using what we call **Armstrong's Axioms**

# ARMSTRONG'S AXIOMS: 1

## Reflexivity

For any subset  $X \subseteq \{A_1, \dots, A_n\}$  :

$$A_1, A_2, \dots, A_n \longrightarrow X$$

- Examples

- $A, B \longrightarrow B$

- $A, B, C \longrightarrow A, B$

- $A, B, C \longrightarrow A, B, C$

# ARMSTRONG'S AXIOMS: 2

## Augmentation

For any attribute sets  $X, Y, Z$  :

if  $X \rightarrow Y$  then  $X, Z \rightarrow Y, Z$

- Examples

- $A \rightarrow B$  implies  $A, C \rightarrow B, C$
- $A, B \rightarrow C$  implies  $A, B, C \rightarrow C$

# ARMSTRONG'S AXIOMS: 3

## Transitivity

For any attribute sets  $X, Y, Z$ :

if  $X \rightarrow Y$  and  $Y \rightarrow Z$  then  $X \rightarrow Z$

- Examples

- $A \rightarrow B$  and  $B \rightarrow C$  imply  $A \rightarrow C$

- $A \rightarrow C, D$  and  $C, D \rightarrow E$  imply  $A \rightarrow E$

# APPLYING ARMSTRONG'S AXIOMS

**Product**(name, category, color, department, price)

1.  $name \rightarrow color$
  2.  $category \rightarrow department$
  3.  $color, category \rightarrow price$
- Infer:  $name, category \rightarrow price$ 
    1. We apply the **augmentation** axiom to (1) to obtain (4)  $name, category \rightarrow color, category$
    2. We apply the **transitivity** axiom to (4), (3) to obtain  $name, category \rightarrow price$

# APPLYING ARMSTRONG'S AXIOMS

**Product**(name, category, color, department, price)

1.  $name \rightarrow color$
  2.  $category \rightarrow department$
  3.  $color, category \rightarrow price$
- Infer:  $name, category \rightarrow color$ 
    1. We apply the **reflexivity** axiom to obtain  
(5)  $name, category \rightarrow name$
    2. We apply the **transitivity** axiom to (5), (1) to obtain  
 $name, category \rightarrow color$

# FD CLOSURE

## FD Closure

If  $F$  is a set of FDs, the **closure**  $F^+$  is the set of all FDs **logically implied** by  $F$

Armstrong's axioms are:

- **sound**: any FD generated by an axiom belongs in  $F^+$
- **complete**: repeated application of the axioms will generate all FDs in  $F^+$

# CLOSURE OF ATTRIBUTE SETS

## Attribute Closure

If  $X$  is an attribute set, the **closure**  $X^+$  is the set of all attributes  $B$  such that:

$$X \rightarrow B$$

In other words,  $X^+$  includes all attributes that are functionally determined from  $X$



# WHY IS CLOSURE NEEDED?

1. Does  $X \rightarrow Y$  hold?
  - we can check if  $Y \subseteq X^+$
2. To compute the **closure**  $F^+$  of FDs
  - for each subset of attributes  $X$ , compute  $X^+$
  - for each subset of attributes  $Y \subseteq X^+$ , output the FD  $X \rightarrow Y$

---

# EXAMPLE

---

**Product**(name, category, color, department, price)

- $name \rightarrow color$
- $category \rightarrow department$
- $color, category \rightarrow price$

## Attribute Closure:

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

# THE CLOSURE ALGORITHM

---

- Let  $X = \{A_1, A_2, \dots, A_n\}$
- **UNTIL**  $X$  doesn't change **REPEAT**:
  - IF**  $B_1, B_2, \dots, B_m \rightarrow C$  is an FD **AND**  
 $B_1, B_2, \dots, B_m$  are all in  $X$
  - THEN** add  $C$  to  $X$
- Output  $X$

# EXAMPLE

---

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

- $A, B \rightarrow C$
- $A, D \rightarrow E$
- $B \rightarrow D$
- $A, F \rightarrow B$

Compute the attribute closures:

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

# KEYS & SUPERKEYS

**superkey**: a set of attributes  $A_1, A_2, \dots, A_n$  such that for any other attribute  $B$  in the relation:

$$A_1, A_2, \dots, A_n \longrightarrow B$$

**key** (or candidate key): a **minimal** superkey

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

If a relation has multiple keys, we specify one to be the **primary key**

# COMPUTING KEYS & SUPERKEYS

- Compute  $X^+$  for all sets of attributes  $X$
- If  $X^+ = \text{all attributes}$ , then  $X$  is a **superkey**
- If no subset of  $X$  is a superkey, then  $X$  is also a key

# EXAMPLE

---

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

- $name \rightarrow color$
- $color, category \rightarrow price$

**Superkeys:**

- $\{name, category\}, \{name, category, price\}$   
 $\{name, category, color\}, \{name, category, price, color\}$

**Keys:**

- $\{name, category\}$

# HOW MANY KEYS?

---

**Q:** Is it possible to have many keys in a relation **R** ?

**YES!!** Take relation **R**(A, B, C) with FDs

- $A, B \longrightarrow C$
- $A, C \longrightarrow B$



---

# RECAP

---

- FDs and (super)keys
- Reasoning with FDs:
  - given a set of FDs, infer all implied FDs
  - given a set of attributes  $X$ , infer all attributes that are functionally determined by  $X$
- Next, we will look at how to use them to detect that a table is “bad”