



# **Database Design: Functional Dependencies**

**Abdu Alawini**

University of Illinois at Urbana-Champaign

CS411: Database Systems

# Overview of Database Design

- Conceptual design: (ER & UML Models are used for this.)
  - What are the **entities** and **relationships** we need?
- Logical design:
  - Transform ER design to Relational Schema
- Schema Refinement: (Normalization)
  - Check relational schema for redundancies and related anomalies.
- Physical Database Design and Tuning:
  - Consider typical workloads; (sometimes) modify the database design; select file types and indexes.

We're here

# Motivation

- We have designed ER diagram, and translated it into a relational db schema  $R = \text{set of } R_1, R_2, \dots$  **Now what?**
- We can do the following
  - implement  $R$  in SQL
  - start using it
- However,  $R$  may not be well-designed, thus causing us a lot of problems
- OR: people may start without an ER diagram, and you need to reformat the schema  $R$ 
  - Either way you may need to **improve** the schema

# How do We Obtain a Good Design?

- Start with the original db schema R
  - From ER translation or otherwise
- Identify its *functional dependencies*
- Use them to transform R until we get a good design  $R^*$

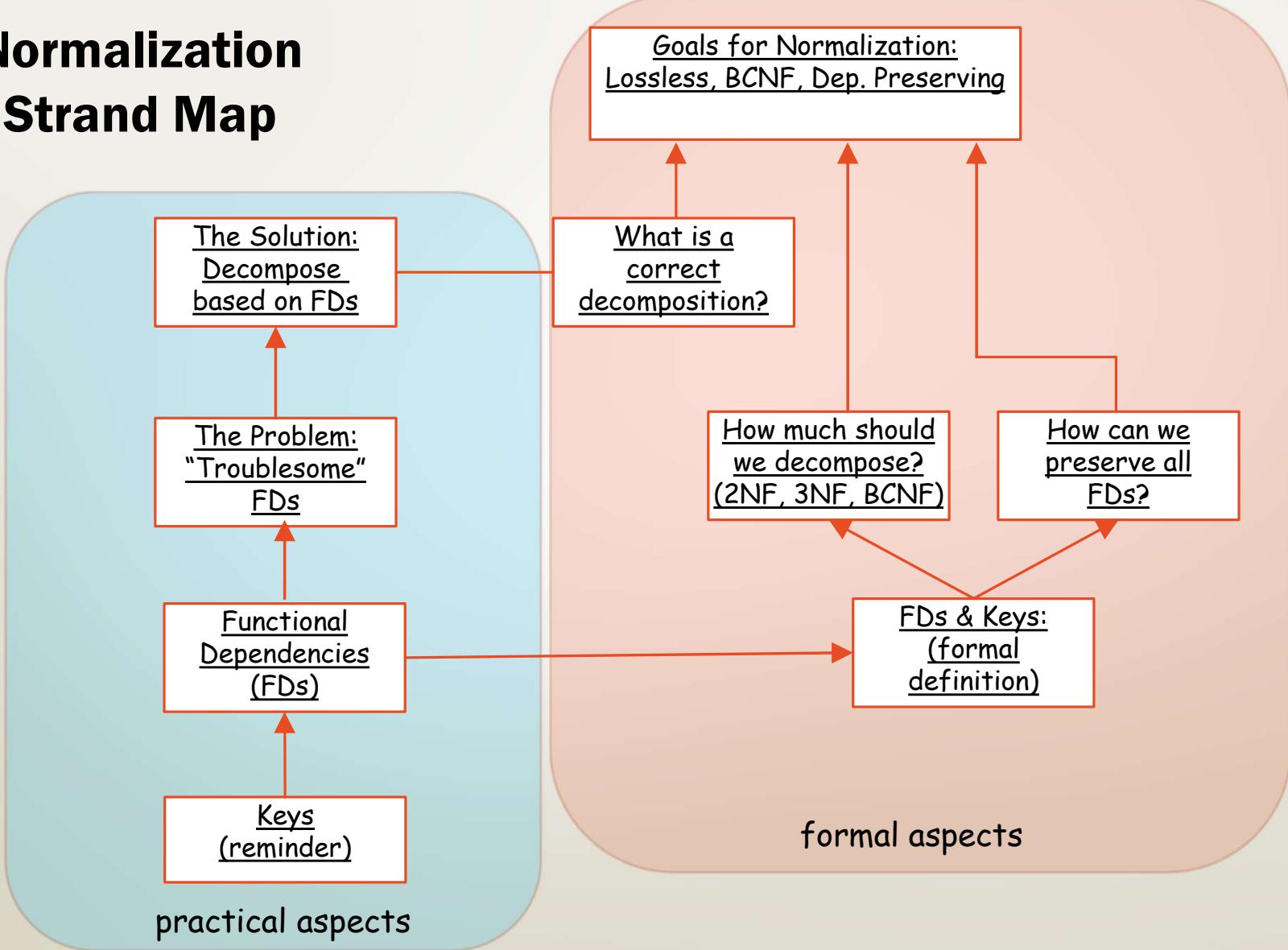
# Desirable Properties of R\*

1. must preserve the information of R (lossless)
2. must have minimal amount of redundancy
3. must be “dependency preserving”
  - (we’ll come to this later)
  - must also give good query performance

# Normal Forms

- DB researchers have developed many “**normal forms**”
- These are basically schemas obeying certain rules
  - Converting a schema that doesn’t obey rules to one that does is called “**normalization**”
  - This typically involves some kind of decomposition into smaller tables.
- (the opposite: grouping tables together, is called “**denormalization**”)

# Normalization Strand Map



# Keys for a Table (reminder)

The key(s) for a table must have unique values and the key(s) for a table help us understand what the table is “about.”

Table Keys				
Account	Number	Owner	Balance	Type
101		J. Smith	1000.00	checking
102		W. Wei	2000.00	checking
103		J. Smith	5000.00	savings
104		M. Jones	1000.00	checking
105		H. Martin	10,000.00	checking

Deposit	AcctNo	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/02/00	10,000.00

Check	AcctNo	Check-number	Date	Amount
	101	924	10/23/00	125.00
	101	925	10/24/00	23.98

## **Notice ... only one value for non-key attributes (for each key value)**

Employee	ssn	name	salary	job-code
	111111111	John Smith	40,000	15
	123456789	Mary Smith	50,000	22
	123456789	Marie Jones	50,000	24

**1. NOT allowed because SSN is key!**

**2. Only one name (and one salary and one job-code) for each row.**

For one particular SSN value, **123-45-6789**, there is only **ONE name** because

1. there is only one tuple and
2. we assume that attributes values are atomic.

# **Functional Dependencies (FDs) generalize keys**

Functional dependencies (FDs) for relational tables are a generalization of the notion of key for a table.

# Functional Dependencies

**Definition:**

If two tuples agree on the attributes

$A_1, A_2 \dots A_n$

then they must also agree on the attributes

$B_1, B_2, \dots B_m$

*Where have we seen  
this before?*

**Formally:**  $A_1, A_2, \dots A_n \longrightarrow B_1, B_2, \dots B_m$

# **Functional Dependencies (from semantics of the application)**

Likely functional dependencies:

$ssn \rightarrow employee-name$

$course-number \rightarrow course-title$

Unlikely functional dependencies

$course-number \not\rightarrow book$

$birthdate \not\rightarrow ssn$

# Will FDs be enforced?

Consider this table:

Emp(ssn, name, phone, dnum, dept-name)



Suppose there is an FD from  $dnum \rightarrow dept\text{-name}$

But  $ssn$  is the key for this table.

What will prevent two names for one dept?

# Will this FD be enforced? Let's try it.

Consider this table:

Emp(ssn, name, phone, dnum, dept-name)

Employee	<u>ssn</u>	Name	Phone	Dnum	Dept name
	111111111	John	555-1234	12	Sales
	222222222	Mary	555-7890	12	Marketing
	...				

Can we put these two rows in this table?

Yes, it doesn't violate the key constraint.

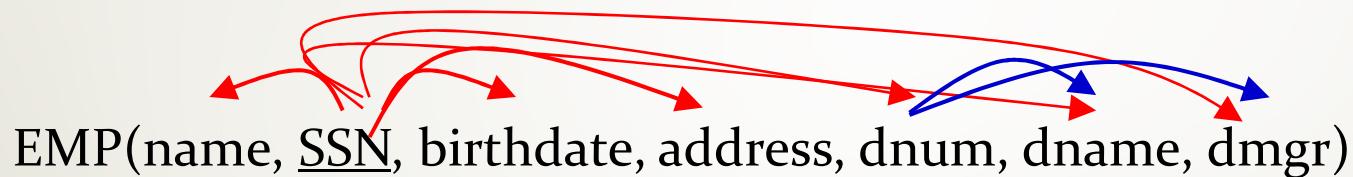
But, the FD from dept to dept-name is violated! We shouldn't have two different names for dnum 12!

# **The Problem: “Troublesome” FDs**

“Troublesome” FDs (FD where the left-hand-side of the FD is NOT a key for the table where its attributes appear) cause redundancy and update anomalies.

# Sometimes Redundancy is Caused by FDs

Consider this table:



Then *dname* and *dmgr* are stored redundantly – whenever there are multiple employees in a department.

This redundancy is caused by what we informally call “troublesome” FDs. The FDs shown in blue are “troublesome”.

# Redundancy Caused by Troublesome FD – Sample Data

EMP(name, <u>SSN</u> , birthdate, address, dnum, <i>dname</i> , <i>dmgr</i> )							
John	111	June 3	123 St.	D1	sales	222	
Sue	222	May 15	455 St.	D1	sales	222	
Max	333	Mar. 5	678 St.	D2	research	333	
Wei	444	May 2	999 St.	D2	research	333	
Tom	555	June 22	888 St.	D2	research	333	

*dname* and *dmgr* are stored redundantly – whenever there are multiple employees in a department.

This redundancy is caused by what we informally call “troublesome” FDs.  
The FDs shown in blue are “troublesome”.

# **Update Anomalies**

## **caused by “troublesome” FDs**

**EMP(name, SSN, birthdate, address, dnum, dname, dmgr)**

**Insertion anomalies:**

if you want to insert a department, you can't ... until  
there is at least one employee.

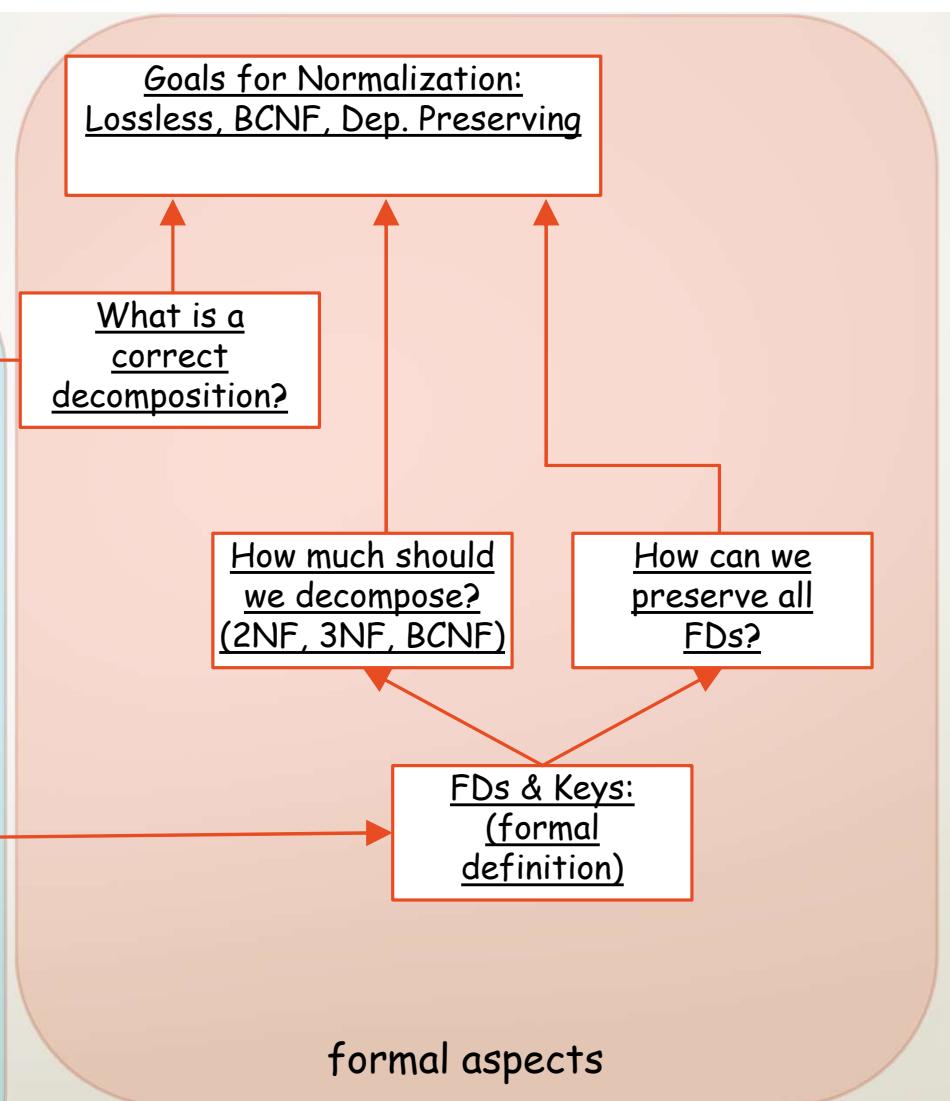
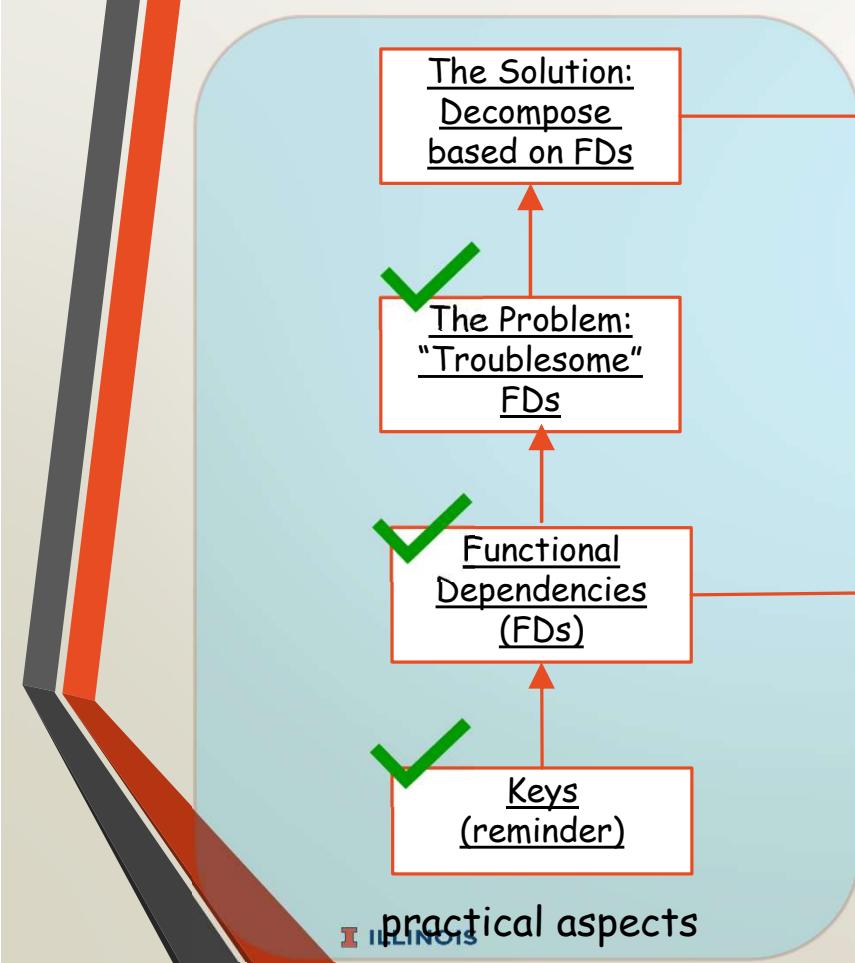
**Deletion anomalies:** if you delete an employee, is that dept.  
gone? Was this the last employee in that dept.?

**Update anomalies:** If you want to change *dname*, for  
example, you need to change it everywhere! And you  
have to find them all first.

Troublesome FDs cause (redundancy and) update  
anomalies.



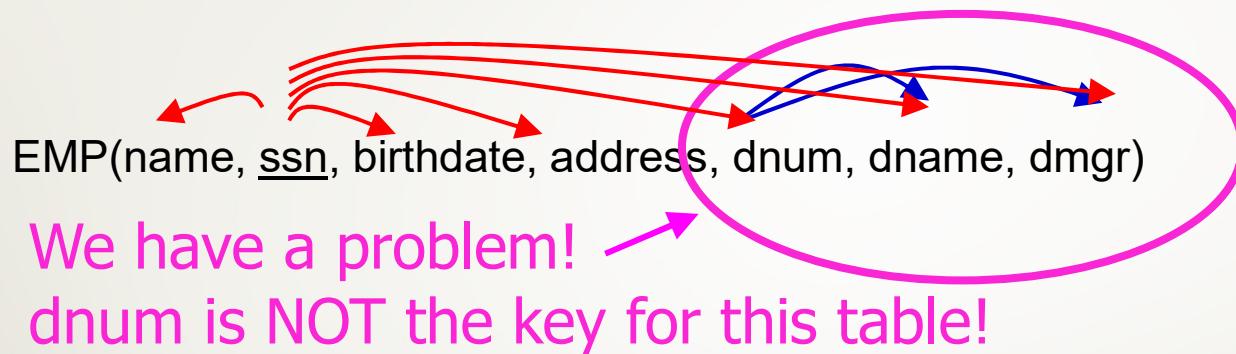
# Normalization Strand Map



# **The Solution: Lifting “Troublesome” FDs**

Normalization by decomposition, based on FDs (where “troublesome” FDs are lifted into a separate table), reduces redundancy and update anomalies.

## Example: Finding Troublesome FDs



So these blue FDs will not be enforced automatically by the DBMS (using only keys).  
And there can be redundancy and update anomalies

# Example: Lifting Troublesome FDs

1. Lift the “troublesome” FD into its own table with dnum as the key. Now they will be enforced.

Dept(dnum, dname, dmgr)

EMP(name, ssn, birthdate, address, dnum, dname, dmgr)

New-Emp(name, ssn, birthdate, address, dnum)

2. Leave the LHS of the “troublesome” FDs behind.

Define a foreign key where

New-Emp.dnum REFERENCES Dept.dnum

# Table is Split onto New Schemas

New-EMP(name, SSN, birthdate, address, dnum)

John	111	June 3	123 St.	D1
Sue	222	May 15	455 St.	D1
Max	333	Mar. 5	678 St.	D2
Wei	444	May 2	999 St.	D2
Tom	555	June 22	888 St.	D2

Insert anomalies: No Problem

Delete anomalies: No Problem

Update anomalies:  
dname is only stored once!

Dept(dnum, dname, dmgr)

D1	sales	222
D2	research	333

*Less redundancy!*

## **Basic Idea: Normalize based on FDs**

- Identify all the (non-trivial) FDs in an application.
  - Identify FDs that are implied by the keys.
  - Identify FDs that are NOT implied by the keys – the “troublesome” ones.
- Decompose a table with a “troublesome” FD into two or more tables by “lifting” each troublesome FD into a table of its own.

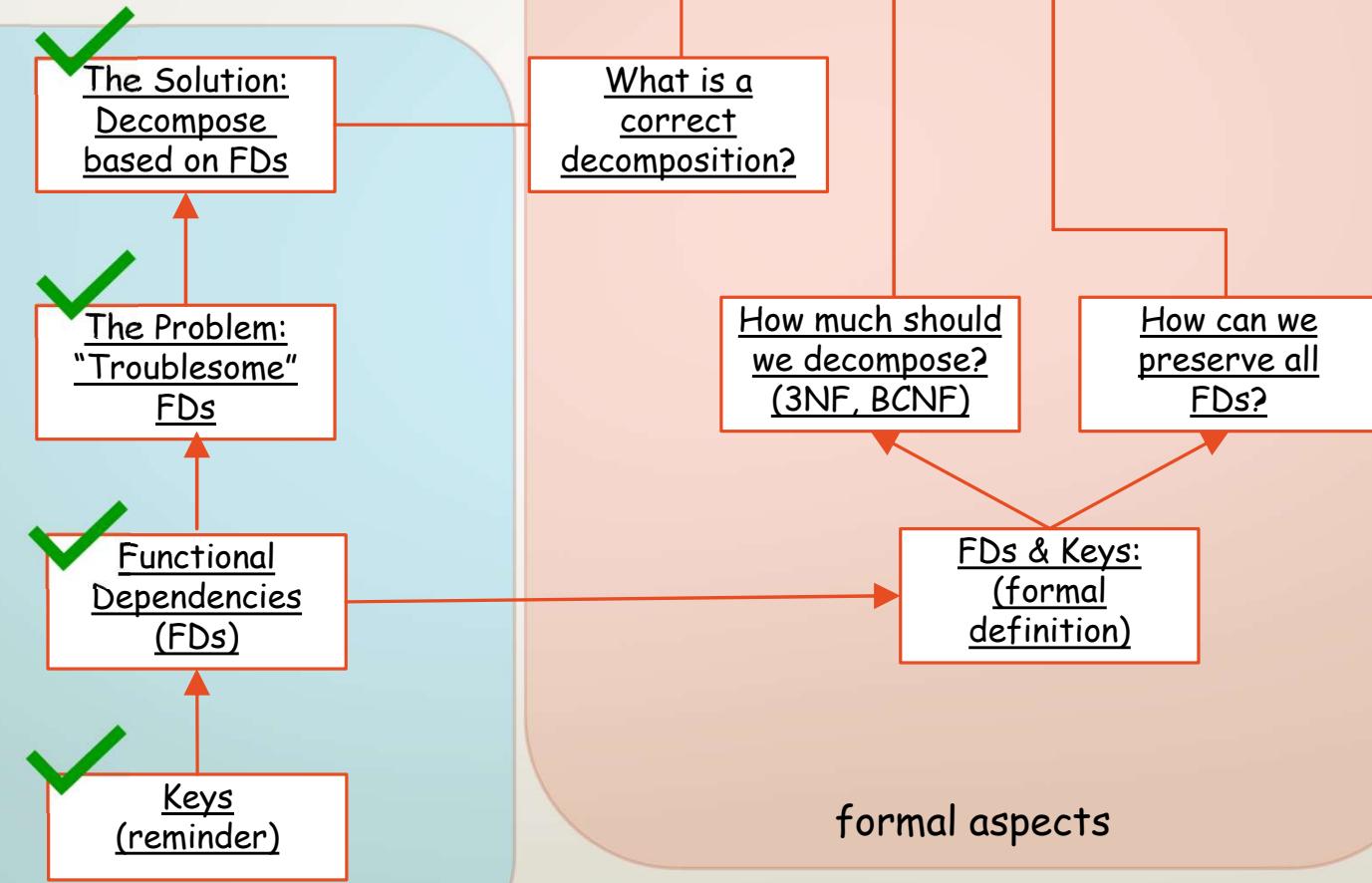
**Note:** when there are two or more “troublesome” FDs with the same left side, then they can be lifted, together, into a single table.

## Questions about normalization

- How do we know which FDs we have?  
*Talk to domain experts; identify FDs; use them as the starting point for normalization.*
- How do we know if the decomposition is correct?
- How do we know how much to normalize?  
How far should we go?



# Normalization Strand Map



practical aspects

A. Alawini

26



# Outline

- Functional Dependencies and Keys
- Inference Rules of Functional Dependencies
- Attribute Closure
- Closure of a Set of FDs

# Reminder: Keys, Candidate Keys, Primary Keys

Reminders:

A **key** is the same as a **candidate key** (synonyms).

If we have two or more keys in a table, we pick one to be the **primary key**.

Example:

([EmpID](#), [SSN](#), Name, Address)

[EmpID](#) is a key (candidate key)

[SSN](#) is a key (candidate key)

We may choose EmpId to be the primary key.

# Definition of a Key for a Relation

A key is a **minimal** set of attributes in a relation whose values are guaranteed to uniquely identify tuples in the relation.

- **minimal?** studentID, studentName is not a key because studentID is a key.
- **minimal?** No subset of the fields that comprise a key is a key
- Two distinct tuples have distinct key values
- Can be more than one key for a table

## **Definition of a Superkey for a Relation**

Every key is (automatically) a superkey.  
A superkey is NOT necessarily a key.

Example:

Emp (SSN, name, phone, dept)

SSN is a key for this relation.

(dept, SSN) is a superkey for this relation (but not a key).

# **Keys and FDs are Constraints**

- We need to know if keys and FDs (always) hold in the application.
- We need to consult a domain expert to find out what the keys and FDs are. The keys and FDs serve as input to the database design process.
- We would like to enforce the keys and FDs constraints.



# Outline

- ✓ Functional Dependencies and Keys
- Inference Rules of Functional Dependencies
- Attribute Closure
- Closure of a Set of FDs

# **Goal: Find ALL Functional Dependencies**

- Anomalies occur when certain “troublesome” FDs hold
- We can identify some FDs
- But we need to find *all* FDs, and then look for the bad ones.

## Inference Rules for FDs

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

Equivalent to:

$$A_1 A_2 \dots A_n \rightarrow B_1;$$

$$A_1 A_2 \dots A_n \rightarrow B_2;$$

...

$$A_1 A_2 \dots A_n \rightarrow B_m$$

Splitting/Combining  
Rule

# Inference Rules for FDs

- $A_1A_2\dots A_n \rightarrow A_1$
- In general,  
 $A_1A_2\dots A_n \rightarrow B_1B_2\dots B_m$

if  $\{B_1B_2\dots B_m\} \subseteq \{A_1A_2\dots A_n\}$

Example: name, UIN  $\rightarrow$  UIN

Trivial Functional  
Dependencies Rule

*Why does this make sense?*

# Inference Rules for FDs

IF

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

AND

$$B_1 B_2 \dots B_m \rightarrow C_1 C_2 \dots C_k$$

THEN

$$A_1 A_2 \dots A_n \rightarrow C_1 C_2 \dots C_k$$

Transitive Closure  
Rule



# Outline

- ✓ Functional Dependencies and Keys
- ✓ Inference Rules of Functional Dependencies
- Attribute Closure
- Closure of a Set of FDs

# Closure of a Set of Attributes

Given a set of attributes  $\{A_1, \dots, A_n\}$  and a set of FDs F.

Problem: find all attributes  $B$  such that:

for all relations that satisfy F, they also satisfy:

$$A_1, \dots, A_n \rightarrow B$$

The **closure** of  $\{A_1, \dots, A_n\}$ , denoted  $\{A_1, \dots, A_n\}^+$ ,  
is the set of all such attributes  $B$

## Example

- Set of attributes A,B,C,D,E,F.
- Functional Dependencies:

A B  $\longrightarrow$  C

A D  $\longrightarrow$  E

B  $\longrightarrow$  D

A F  $\longrightarrow$  B

Closure of  $\{A, B\}^+$ :

Closure of  $\{A, F\}^+$  :

## Example

- Set of attributes A,B,C,D,E,F.
- Functional Dependencies:

A B  $\longrightarrow$  C

A D  $\longrightarrow$  E

B  $\longrightarrow$  D

A F  $\longrightarrow$  B

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

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

# Algorithm to Compute Closure

Split the FDs in F so that every FD has a single attribute on the right. (Simplify the FDs)

Start with  $X = \{A_1 A_2 \dots A_n\}$ .

**Repeat until X doesn't change do:**

If  $(B_1 B_2 \dots B_m \rightarrow C)$  is in F,  
such that  $B_1, B_2, \dots, B_m$  are in X and C is not in X:  
add C to X.

// X is now the correct value of  $\{A_1 A_2 \dots A_n\}^+$

*Why does this algorithm converge?*

# Uses for Attribute Closure

- Use 1: To test if  $X$  is a (super)key
  - **How?** By computing  $X^+$ , and check if  $X^+$  contains all attrs of  $R$
  - We can also use it to find candidate keys
    - Compute  $X^+$  for all sets  $X$  where  $X^+ = \text{all attributes}$
    - Then list only the minimal  $X$ 's
- Use 2: To check if  $X \rightarrow Y$  holds
  - **How?** By checking if  $Y$  is contained in  $X^+$

# Finding Keys Example 1

Given  $R(A, B, C, D, E, F)$  and  $F = \{A \rightarrow CD, D \rightarrow E, A \rightarrow B\}$

What are the candidate keys?

<b>LEFT:</b> Attributes that only appear in the LHS of any FD	<b>MIDDLE:</b> Attributes that appear in the LHS of some FDs and in the RHS of others	<b>RIGHT:</b> Attributes that only appear in the RHS of any FD	<b>NONE:</b> Attributes that do not appear in any FD
A	D	B,C,E	F

- Every attribute that appears in the LEFT and NONE columns must be part of any candidate key
- Compute the attribute closure of the LEFT+ NONE attributes:  $AF^+ = \{A, B, C, D, E, F\}$ 
  - if all attributes are included, then your key is LEFT + NONE: {A,F}
  - Otherwise, try combinations of all LEFT attributes with subset of the MIDDLE attributes

AF is the only Candidate Keys for R

## Finding Keys Example 2

Given R(A, B, C, D, E, F) and F= {B->D, C->AE, B->F, A->C)

What are the candidate keys?

LEFT	MIDDLE	RIGHT	NONE
B	A,C	D,E,F	

- Compute the attribute closure of the LEFT attributes:  $B^+ = \{B, D, F\}$
- We can't get to all attributes from B, so we must try adding subsets of MIDDLE to LEFT.
  - $BC^+ = \{A, B, C, D, E, F\}$ , we can get to all attributes, so BC is a candidate key
  - $AB^+ = \{A, B, C, D, E, F\}$ , we can get to all attributes, so AB is also a candidate key

Two Candidate Keys for R: AB, BC



# Outline

- ✓ Functional Dependencies and Keys
- ✓ Inference Rules of Functional Dependencies
- ✓ Attribute Closure
- Closure of a Set of FDs

# Closure of a set of FDs

- Given a relation schema R & a set F of FDs
  - Closure of F:  $F^+$  = all FDs logically implied by F
  - Allows us to answer all questions of the type
    - is the FD  $f$  logically implied by F?
- Example
  - $R = \{A, B, C, G, H, I\}$
  - $F = A \rightarrow B; A \rightarrow C; CG \rightarrow H; CG \rightarrow I; B \rightarrow H$
  - would  $A \rightarrow H$  be logically implied?
    - yes (you can prove this, using the definition of FD)
- How to compute  $F^+?$



# Using Attribute Closure to Infer ALL FDs

Example:

Given  $R(A, B, C, D)$  and

$F = \{AB \rightarrow C; AD \rightarrow B; B \rightarrow D\}$

Compute the set of all inferred FDs of  $F$

Step 1: Computer  $X^+$ , for every  $X$ :

$A^+ = A, B^+ = BD, C^+ = C, D^+ = D$

$AB^+ = ABCD, AC^+ = AC, AD^+ = ABCD, BC^+ = BCD, BD^+ = BD, CD^+ = CD$

$ABC^+ = ABD^+ = ACD^+ = ABCD$  (no need to compute, why?)

$BCD^+ = BCD, ABCD^+ = ABCD$

Step 2: Enumerate all FD's  $X \rightarrow Y$ , s.t.  $Y \subseteq X^+$  and  $X \cap Y = \emptyset$

$B \rightarrow D, AB \rightarrow CD, AD \rightarrow BC, BC \rightarrow D, ABC \rightarrow D, ABD \rightarrow C, ACD \rightarrow B$

# Normalization Strand Map

