

# Database Design: Normalization

# Agenda

1. Database Design
2. Normal forms & functional dependencies
3. Finding functional dependencies
4. Closures, superkeys & keys

# Design Theory

- The biggest problem needed to be solved in database is *data redundancy*.
- Why data redundancy is the problem? Because it causes:
  - Insert Anomaly
  - Update Anomaly
  - Delete Anomaly
- Design theory is about how to represent your data to avoid *anomalies*.
- Achieved by **Data Normalization**, a process of analyzing a relation to ensure that it is well formed.
- Normalization involves **decomposing** relations with anomalies to produce smaller well structured relations.
- If a relation is normalized (or well formed), rows can be **inserted, deleted and modified** without creating anomalies.

# Data Anomalies & Constraints

## Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes  
***anomalies***:

Student	Course	Room
Mary	CSC261	101
Joe	CSC261	101
Sam	CSC261	101
..	..	..

If every course is in only one room,  
contains **redundant** information!

## Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes  
***anomalies***:

Student	Course	Room
Mary	CSC261	101
Joe	CSC261	703
Sam	CSC261	101
..	..	..

If we update the room number for one tuple, we get inconsistent data = an **update anomaly**

## Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes  
***anomalies:***

Student	Course	Room
..	..	..

If everyone drops the class, we lose what  
room the class is in! = a **delete anomaly**


## Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes ***anomalies***:

Student	Course	Room
Mary	CSC261	B01
Joe	CSC261	B01
Sam	CSC261	B01
..	..	..

...	CSC461	703
-----	--------	-----



Similarly, we can't reserve a room without students = an ***insert anomaly***



## Constraints Prevent (some) Anomalies in the Data

Student	Course
Mary	CSC261
Joe	CSC261
Sam	CSC261
..	..

Course	Room
CSC261	101
CSC257	601

Is this form better?

- Redundancy?
- Update anomaly?
- Delete anomaly?
- Insert anomaly?

Today: develop theory to understand why this design may be better **and** how to find this *decomposition*...

# Database Anomalies

## Example 2

Anomalies are problems caused by **bad database design**.

**Example:**

ACTIVITY(StudentID, Activity, Fee)

ACTIVITY Relation

StudentID	Activity	Fee
100	Skiing	200
100	Golf	65
175	Squash	50
175	Swimming	50
200	Swimming	50
200	Golf	65

An **insertion anomaly** occurs when a row cannot be added to a relation, because not all data are available (or one has to invent “dummy” data)

- ❖ **Example:** we want to store that scuba diving costs \$175, but have no place to put this information until a student takes up scuba-diving (unless we create a fake student)

A **deletion anomaly** occurs when data is deleted from a relation, and other critical data are unintentionally lost

- ❖ **Example:** if we delete the record with StudentID = 100, we forget that skiing costs \$200

An **update anomaly** occurs when one must make many changes to reflect the modification of a single datum

- ❖ **Example:** if the cost of swimming changes, then all entries with swimming Activity must be changed too

# Cause of Anomalies

Anomalies are primarily **caused** by:

1. **Data redundancy**: replication of the same field in multiple tables, other than foreign keys
2. **Functional dependencies** including:
  - Partial dependency
  - Transitive dependency
  - Multi-value dependency

# Functional Dependencies

# Functional Dependencies for Dummies

- A **relationship** between **attributes** where one attribute (or group of attributes) **determines** the value of another attribute (or group of attributes) in the **same** table.
- **Example:**  
SSN uniquely identify any Person

(SSN)  $\rightarrow$  (First Name, Last Name)

# Candidate Keys/Primary Keys and Functional Dependencies

By definition:

- A **candidate key** of a relation **functionally determines** all other **non-key** attributes in the row.

Implies:

- A **primary key** of a relation **functionally determines** all other non-key attributes in the row.

EmployeeID → (EmployeeName, EmpPhone)

# Functional Dependency

**Def:** Let  $A, B$  be sets of attributes, we write  $A \rightarrow B$  or say  $A$  **functionally determines**  $B$  if, for any tuples  $t_1$  and  $t_2$ :

$t_1[A] = t_2[A]$  implies  $t_1[B] = t_2[B]$  and we

call  $A \rightarrow B$  a **functional dependency**

*$A \rightarrow B$  means that*

*“whenever two tuples agree on  $A$  then they agree on  $B$ .”*

$A$	It is a <b>determinant</b> set.
$B$	It is a <b>dependent</b> attribute.
$\{A \rightarrow B\}$	$A$ <b>functionally determines</b> $B$ . $B$ is a functionally dependent on $A$ .

# A Picture of FDs

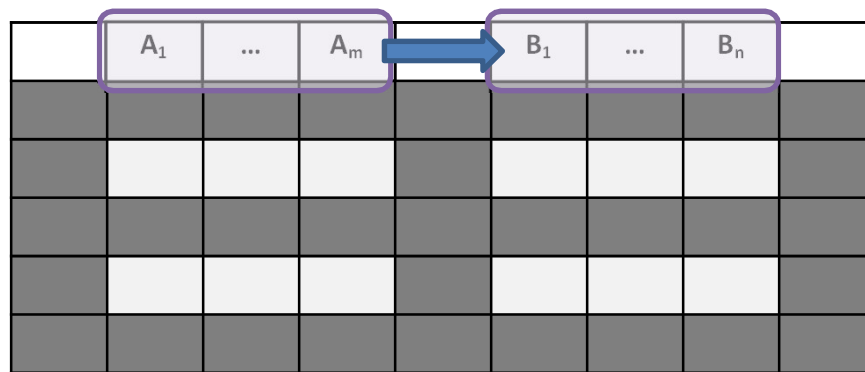
	A <sub>1</sub>	...	A <sub>m</sub>		B <sub>1</sub>	...	B <sub>n</sub>	

Defn (again):

Given attribute sets  $\mathbf{A} = \{A_1, \dots, A_m\}$   
and  $\mathbf{B} = \{B_1, \dots, B_n\}$  in  $\mathbf{R}$ ,



# A Picture of FDs



Defn (again):

Given attribute sets  $A = \{A_1, \dots, A_m\}$   
and  $B = \{B_1, \dots, B_n\}$  in  $R$ ,

The *functional dependency*  $A \rightarrow B$   
**on**  $R$  holds if for *any*  $t_i, t_j$  in  $R$ :

# A Picture of FDs

	A <sub>1</sub>	...	A <sub>m</sub>		B <sub>1</sub>	...	B <sub>n</sub>	
t <sub>i</sub>								
t <sub>j</sub>								

If t<sub>1</sub>, t<sub>2</sub> agree here..

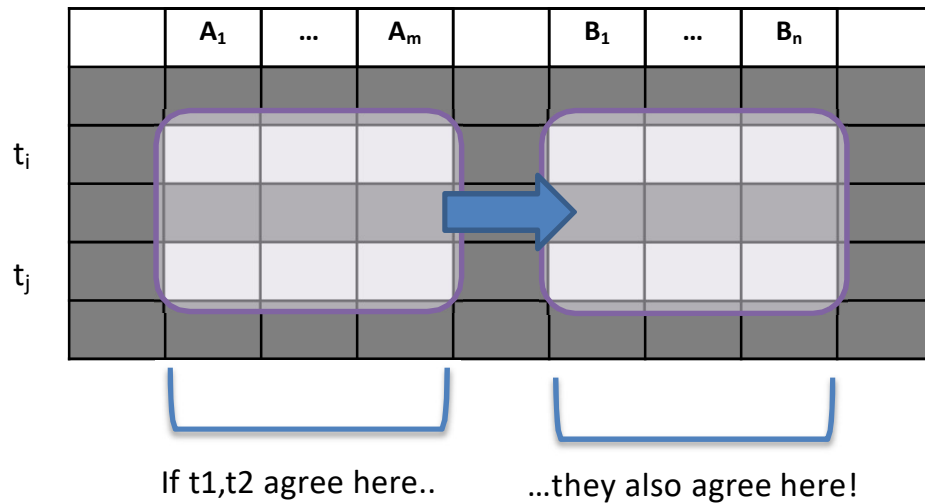
Defn (again):

Given attribute sets  $A = \{A_1, \dots, A_m\}$   
and  $B = \{B_1, \dots, B_n\}$  in  $R$ ,

The **functional dependency**  $A \rightarrow B$   
**on**  $R$  holds if for **any**  $t_i, t_j$  in  $R$ :

$t_i[A_1] = t_j[A_1]$  AND  $t_i[A_2] = t_j[A_2]$  AND ...  
AND  $t_i[A_m] = t_j[A_m]$

# A Picture of FDs



Defn (again):

Given attribute sets  $\mathbf{A} = \{A_1, \dots, A_m\}$   
and  $\mathbf{B} = \{B_1, \dots, B_n\}$  in  $R$ ,

The **functional dependency**  $\mathbf{A} \rightarrow \mathbf{B}$   
**on**  $R$  holds if for **any**  $t_i, t_j$  in  $R$ :

**if**  $t_i[A_1] = t_j[A_1]$  AND  $t_i[A_2] = t_j[A_2]$  AND  
... AND  $t_i[A_m] = t_j[A_m]$

**then**  $t_i[B_1] = t_j[B_1]$  AND  $t_i[B_2] = t_j[B_2]$   
AND ... AND  $t_i[B_n] = t_j[B_n]$

# FDs for Relational Schema Design

High-level idea: **why do we care about FDs?**

1. Start with some relational **schema** (e.g., design by ER diagram)
2. Find out its **functional dependencies** (FDs)
3. Use these to **design a better schema**
  - One which minimizes the possibility of anomalies

# Functional Dependencies as Constraints

A **functional dependency** is a form of **constraint**

- *Holds* on some instances not others.
- Part of the schema, helps define a valid *instance*.

Recall: an ***instance*** of a schema is a multiset of tuples conforming to that schema, **i.e. a table**

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

Note: The FD  
 $\{\text{Course}\} \rightarrow \{\text{Room}\}$  ***holds on this instance***

# Functional Dependencies as Constraints

Note that:

- 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*

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

However, cannot *prove* that the FD {Course} -> {Room} is ***part of the schema***

# More Examples

An FD is a constraint which 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

# More Examples

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

{Position} → {Phone}



# More Examples

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

but *not* {Phone} → {Position}

# ACTIVITY

A	B	C	D	E
1	2	4	3	6
3	2	5	1	8
1	4	4	5	7
1	2	4	3	6
3	2	5	1	8

Find at least *three* FDs which hold on this instance:

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

# Armstrong inference rules

- Armstrong's Axioms is a set of rules.
- It provides a simple technique for reasoning about functional dependencies.
- It was developed by William W. Armstrong in 1974.
- It is used to infer all the functional dependencies on a relational database.

# Armstrong inference rules

## A. Primary Rules:

- |               |   |
|---------------|---|
| <b>Rule 1</b> | <b>Reflexivity</b><br>If A is a set of attributes and B is a subset of A, then A holds B. $\{ A \rightarrow B \}$<br>(If $B \subseteq A$ , then $A \rightarrow B$ )   |
| <b>Rule 2</b> | <b>Augmentation</b><br>If A hold B and C is a set of attributes, then AC holds BC. $\{ AC \rightarrow BC \}$<br>(If $A \rightarrow B$ , then $AC \rightarrow BC$ )<br>It means that attribute in dependencies does not change the basic dependencies. |
| <b>Rule 3</b> | <b>Transitivity</b><br>If A holds B and B holds C, then A holds C.<br>If $\{ A \rightarrow B \}$ and $\{ B \rightarrow C \}$ , then $\{ A \rightarrow C \}$<br>A holds B $\{ A \rightarrow B \}$ means that A functionally determines B.              |

# Armstrong inference rules

## B. Secondary Rules:

### Union

**Rule 1**

If A holds B and A holds C, then A holds BC.  
If  $\{A \rightarrow B\}$  and  $\{A \rightarrow C\}$ , then  $\{A \rightarrow BC\}$

### Decomposition

**Rule 2**

If A holds BC and A holds B, then A holds C.  
If  $\{A \rightarrow BC\}$  and  $\{A \rightarrow B\}$ , then  $\{A \rightarrow C\}$

### Pseudo Transitivity

**Rule 3**

If A holds B and BC holds D, then AC holds D.  
If  $\{A \rightarrow B\}$  and  $\{BC \rightarrow D\}$ , then  $\{AC \rightarrow D\}$

# Armstrong inference rules

## ***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  $X \rightarrow Z$ , then  $X \rightarrow YZ$

Decomposition: if  $X \rightarrow YZ$ , then  $X \rightarrow Y$  and  $X \rightarrow Z$

Pseudo transitivity: if  $X \rightarrow Y$  and  $WY \rightarrow Z$ , then  $XW \rightarrow Z$

# Armstrong inference rules

Axioms are both

***Sound:***

when applied to a set of functional dependencies they only produce dependency tables that belong to the transitive closure of that set

***Complete:***

can produce all dependency tables that belong to the transitive closure of the set

# Armstrong inference rules

Three last rules can be derived from the first three (the axioms)

Let us look at the ***union rule***:

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

Using the first three axioms, we have:

if  $X \rightarrow Y$ , then  $XX \rightarrow XY$  same as  $X \rightarrow XY$  (2<sup>nd</sup>)

if  $X \rightarrow Z$ , then  $YX \rightarrow YZ$  same as  $XY \rightarrow YZ$  (2<sup>nd</sup>)

if  $X \rightarrow XY$  and  $XY \rightarrow YZ$ , then  $X \rightarrow YZ$  (3<sup>rd</sup>)



**Example:**

Consider relation  $E = (P, Q, R, S, T, U)$  having set of Functional Dependencies (FD).

$P \rightarrow Q$	$P \rightarrow R$
$QR \rightarrow S$	$Q \rightarrow T$
$QR \rightarrow U$	$PR \rightarrow U$

**Calculate some members of axioms are as follows:**

1.  $P \rightarrow T$
2.  $PR \rightarrow S$
3.  $QR \rightarrow SU$
4.  $PR \rightarrow SU$

***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  $X \rightarrow Z$ , then  $X \rightarrow YZ$

Decomposition: if  $X \rightarrow YZ$ , then  $X \rightarrow Y$  and  $X \rightarrow Z$

Pseudo transitivity: if  $X \rightarrow Y$  and  $WY \rightarrow Z$ , then  $XW \rightarrow Z$

## Solution:

### 1. $P \rightarrow T$

In the FD set,  $P \rightarrow Q$  and  $Q \rightarrow T$

**So, Using Transitive Rule: If  $\{A \rightarrow B\}$  and  $\{B \rightarrow C\}$ , then  $\{A \rightarrow C\}$**

$\therefore$  If  $P \rightarrow Q$  and  $Q \rightarrow T$ , then  $P \rightarrow T$ .

### 2. $PR \rightarrow S$

In the above FD set,  $P \rightarrow Q$

As,  $QR \rightarrow S$

**So, Using Pseudo Transitivity Rule: If  $\{A \rightarrow B\}$  and  $\{BC \rightarrow D\}$ , then  $\{AC \rightarrow D\}$**

$\therefore$  If  $P \rightarrow Q$  and  $QR \rightarrow S$ , then  $PR \rightarrow S$ .

### 3. $QR \rightarrow SU$

In above FD set,  $QR \rightarrow S$  and  $QR \rightarrow U$

**So, Using Union Rule: If  $\{A \rightarrow B\}$  and  $\{A \rightarrow C\}$ , then  $\{A \rightarrow BC\}$**

$\therefore$  If  $QR \rightarrow S$  and  $QR \rightarrow U$ , then  $QR \rightarrow SU$ .

### 4. $PR \rightarrow SU$

**So, Using Pseudo Transitivity Rule: If  $\{A \rightarrow B\}$  and  $\{BC \rightarrow D\}$ , then  $\{AC \rightarrow D\}$**

$\therefore$  If  $PR \rightarrow S$  and  $PR \rightarrow U$ , then  $PR \rightarrow SU$ .

#### **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  $X \rightarrow Z$ , then  $X \rightarrow YZ$

Decomposition: if  $X \rightarrow YZ$ , then  $X \rightarrow Y$  and  $X \rightarrow Z$

Pseudo transitivity: if  $X \rightarrow Y$  and  $WY \rightarrow Z$ , then  $XW \rightarrow Z$

# Trivial Functional Dependency

## Trivial

If A holds B  $\{A \rightarrow B\}$ , where B is a subset of A, then it is called a **Trivial Functional Dependency**. Trivial always holds Functional Dependency.

## Non-Trivial

If A holds B  $\{A \rightarrow B\}$ , where B is **not** a subset A, then it is called as a **Non-Trivial Functional Dependency**.

# Normalization

<https://www.youtube.com/watch?v=UrYLYV7WSHM>  
<https://www.youtube.com/watch?v=l5DCnCzDb8g>

# Normalization

- **Normalization** is the process of removing **redundant** data from your tables to improve storage efficiency, data integrity, and scalability.
- Normalization generally involves **splitting** existing tables into multiple ones, which must be re-joined or linked each time a query is issued.
- Why normalization?
  - The relation derived from the user view or data store will most likely be **unnormalized**.
  - The problem usually happens when an existing system uses unstructured file, e.g. in MS Excel.



# Unnormalized Form (table)

## Example

ClientRental

clientNo	cName	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	John Kay	PG4	6 Lawrence St, Glasgow	1-Jul-03	31-Aug-04	350	CO40	Tina Murphy
		PG16	5 Novar Dr, Glasgow	1-Sep-04	1-Sep-05	450	CO93	Tony Shaw
CR56	Aline Stewart	PG4	6 Lawrence St, Glasgow	1-Sep-02	10-June-03	350	CO40	Tina Murphy
		PG36	2 Manor Rd, Glasgow	10-Oct-03	1-Dec-04	375	CO93	Tony Shaw
		PG16	5 Novar Dr, Glasgow	1-Nov-05	10-Aug-06	450	CO93	Tony Shaw

# Normalization Example

- (Student ID)  $\rightarrow$  (Student Name, DormName, DormCost)
- However, if
  - (DormName)  $\rightarrow$  (DormCost)

Then, DormCost should be put into its own relation, resulting in:

(Student ID)  $\rightarrow$  (Student Name, DormName)

(DormName)  $\rightarrow$  (DormCost)

# Normalization Example

- (AttorneyID, ClientID)  $\rightarrow$  (ClientName, MeetingDate, Duration)
- However, if
  - ClientID  $\rightarrow$  ClientName
- Then: ClientName should be in its own relation:
- (AttorneyID, ClientID)  $\rightarrow$  (MeetingDate, Duration)
- (ClientID)  $\rightarrow$  (ClientName)



## Steps of Normalization

- ✓ First Normal Form (1NF)
- ✓ Second Normal Form (2NF)
- ✓ Third Normal Form (3NF)
- ✓ Boyce-Codd Normal Form (BCNF)
- ✓ Fourth Normal Form (4NF)
- ✓ Fifth Normal Form (5NF)
- ✓ Domain Key Normal Form (DKNF)

In practice, 1NF, 2NF, 3NF, and BCNF are enough for database.



# Normal Forms

- 1<sup>st</sup> Normal Form (1NF) = All tables are flat

- 2<sup>nd</sup> Normal Form (2NF)

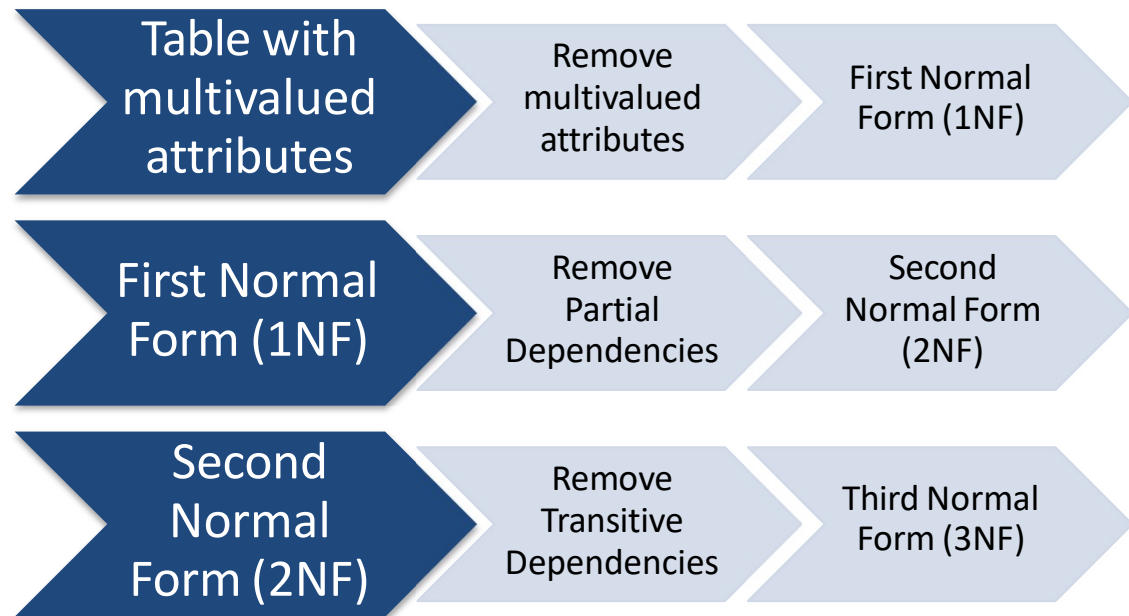
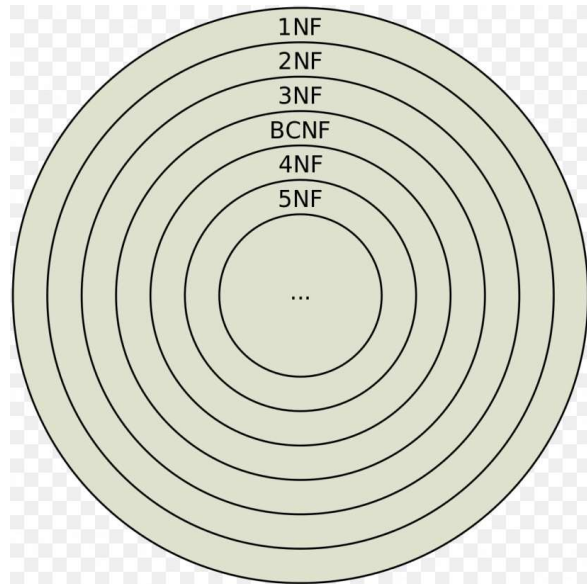
- 3<sup>rd</sup> Normal Form (3NF)

- Boyce-Codd Normal Form (BCNF)

DB designs based on  
*functional dependencies*,  
intended to prevent data  
*anomalies*

- 4<sup>th</sup> and 5<sup>th</sup> Normal Forms = see text books

# Normalization Steps



## First Normal Form (1NF)

The official qualifications for 1NF are:

1. Each **attribute name** must be unique.
2. Each **attribute value** must be single.
3. Each **row** must be unique.
4. There is **no repeating groups**.

Additional:

Choose a primary key.

Reminder:

A primary key is ***unique, not null, unchanged***. A primary key can be either an attribute or combined attributes.



## 1<sup>st</sup> Normal Form (1NF)

Student	Courses
Mary	{CS145, CS229}
Joe	{CS145, CS106}
...	...

***Violates 1NF.***

Student	Courses
Mary	CS145
Mary	CS229
Joe	CS145
Joe	CS106

In 1<sup>st</sup> NF

**1NF Constraint: Types must be atomic!**

## First Normal Form (1NF) (Cont.)

Example of a table not in 1NF :

Group	Topic	Student	Score	
Group A	Intro MongoDB	Sok San	18	marks
		Sao Ry	17	marks
Group B	Intro MySQL	Chan Tina	19	marks
		Tith Sophea	16	marks

It violates the 1NF because:

- Attribute values are not single.
- Repeating groups exists.



## First Normal Form (1NF) (Cont.)

➤ After eliminating:

Group	Topic	Family Name	Given Name	Score
A	Intro MongoDB	Sok	San	18
A	Intro MongoDB	Sao	Ry	17
B	Intro MySQL	Chan	Tina	19
B	Intro MySQL	Tith	Sophea	16

➤ Now it is in 1NF.  
However, it might still violate 2NF and so on.




# Functional Dependencies

We say an attribute, B, has a *functional dependency* on another attribute, A, if for any two records, which have the same value for A, then the values for B in these two records must be the same. We illustrate this as:

$A \rightarrow B$  (read as: *A determines B or B depends on A*)

Employee_name	Project	Email_address
Joe San	POS Mart Sys	<a href="mailto:soksan@yahoo.com">soksan@yahoo.com</a>
Sao Ry	Univ Mgt Sys	<a href="mailto:sao@yahoo.com">sao@yahoo.com</a>
Joe San	Web Redesign	<a href="mailto:soksan@yahoo.com">soksan@yahoo.com</a>
Chan Sokna	POS Mart Sys	<a href="mailto:chan@gmail.com">chan@gmail.com</a>
Sao Ry	DB Design	<a href="mailto:sao@yahoo.com">sao@yahoo.com</a>

Employee\_name  $\rightarrow$  Email\_address



---



## Functional Dependencies (cont.)

<u>EmpNum</u>	EmpEmail	EmpFname	EmpLname
123	<a href="mailto:jdoe@abc.com">jdoe@abc.com</a>	John	Doe
456	<a href="mailto:psmith@abc.com">psmith@abc.com</a>	Peter	Smith
555	<a href="mailto:alee1@abc.com">alee1@abc.com</a>	Alan	Lee
633	<a href="mailto:pdoe@abc.com">pdoe@abc.com</a>	Peter	Doe
787	<a href="mailto:alee2@abc.com">alee2@abc.com</a>	Alan	Lee

If EmpNum is the PK then the FDs:

**EmpNum  $\rightarrow$  EmpEmail, EmpFname, EmpLname**

must exist.

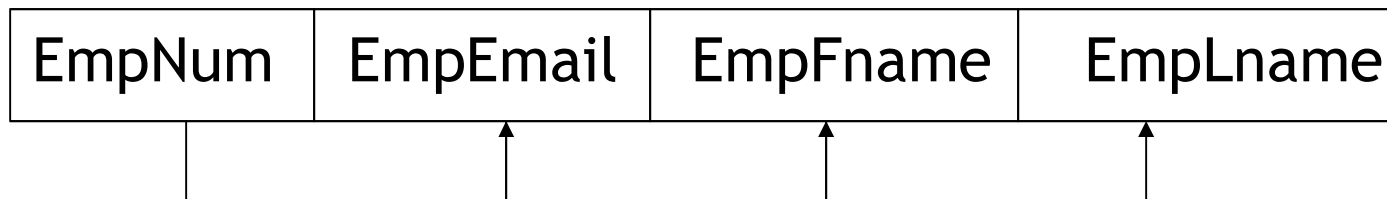
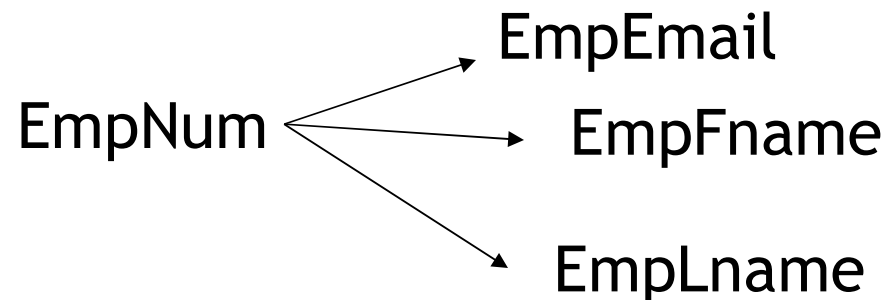


-----

## Functional Dependencies (cont.)

EmpNum  $\rightarrow$  EmpEmail, EmpFname, EmpLname

*3 different ways  
you might see  
FDs depicted*



# Determinant

## Functional Dependency

$\text{EmpNum} \rightarrow \text{EmpEmail}$

Attribute on the left hand side is known as the ***determinant***

- EmpNum is a ***determinant*** of EmpEmail



## Second Normal Form (2NF)

The official qualifications for 2NF are:

1. A table is already in 1NF.
2. All non-key attributes are fully dependent on the primary key.

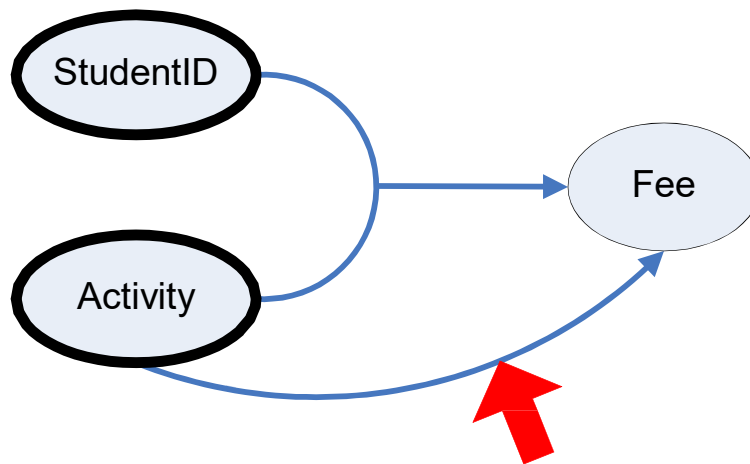
All **partial dependencies** are removed to place in another table.



# Partial Dependencies

- Partial dependency is a functional dependency whose determinant is part of the primary key (but not all of it)
- Example:

ACTIVITY(StudentID, Activity, Fee)



<u>StudentID</u>	<u>Activity</u>	Fee
100	Skiing	200
100	Golf	65
175	Squash	50
175	Swimming	50
200	Swimming	50
200	Golf	65

Example of a table not in 2NF:

<b>CourseID</b>	<b>SemesterID</b>	<b>Num Student</b>	<b>Course Name</b>
IT101	201301	25	Database
IT101	201302	25	Database
IT102	201301	30	Web Prog
IT102	201302	35	Web Prog
IT103	201401	20	Networking

Primary Key

The *Course Name* depends on only *CourseID*, a part of the primary key not the whole primary {*CourseID*, *SemesterID*}. It's called **partial dependency**.

### **Solution:**

*Remove **CourseID** and **Course Name** together to create a new table.*



CourseID	Course Name
IT101	Database
IT101	Database
IT102	Web Prog
IT102	Web Prog
IT103	Networking

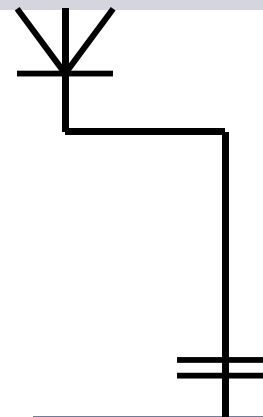
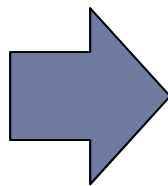
<u>CourseID</u>	<u>SemesterID</u>	Num Student
IT101	201301	25
IT101	201302	25
IT102	201301	30
IT102	201302	35
IT103	201401	20

Done?

Oh no, it is still not in 1NF yet.

Remove the repeating groups too.

Finally, connect the relationship.



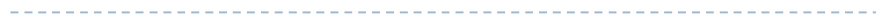
<u>CourseID</u>	Course Name
IT101	D atabase
IT102	Web Prog
IT103	Networking

## Third Normal Form (3NF)

The official qualifications for 3NF are:

1. A table is already in 2NF.
2. Nonprimary key attributes do not depend on other nonprimary key attributes  
(i.e. no **transitive dependencies**)

All transitive dependencies are removed to place in another table.





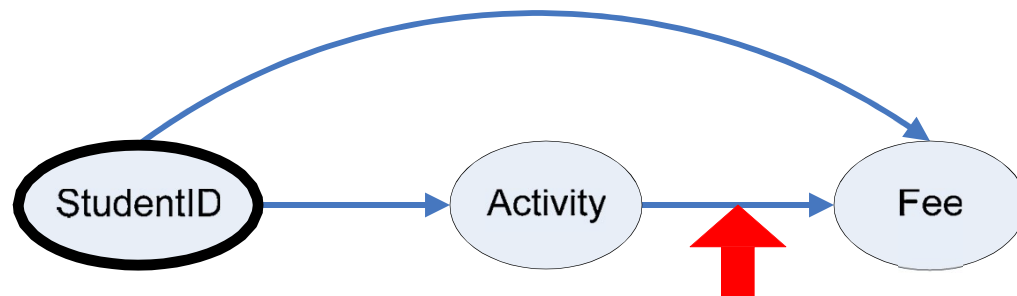
# Transitive Dependencies

Transitive dependency is a functional dependency whose determinant is not the primary key, part of the primary key, or a candidate key.

Transitive functionality is a functional dependency in which a non-key attribute is determined by another non-key attribute.

Example:

ACTIVITY(StudentID, Activity, Fee)




<u>StudentID</u>	Activity	Fee
100	Skiing	200
100	Golf	65
175	Squash	50
175	Swimming	50
200	Swimming	50
200	Golf	65

Example of a Table not in 3NF:

<u>StudyID</u>	<u>CourseName</u>	TeacherName	TeacherTel
1	Database	Sok Piseth	012 123 456
2	Database	Sao Kanha	0977 322 111
3	Web Prog	Chan Veasna	012 412 333
4	Web Prog	Chan Veasna	012 412 333
5	Networking	Pou Sambath	077 545 221



Primary Key



The *TeacherTel* is a nonkey attribute, and the *TeacherName* is also a nonkey attribute. But *TeacherTel* depends on *TeacherName*. It is called **transitive dependency**.

### Solution:

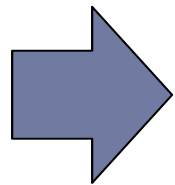
*Remove Teacher Name and TeacherTel together to create a new table.*



Teacher Name	Teacher Tel
Sok Piseth	012 123 456
Sao Kanha	0977 322 111
Chan Veasna	012 412 333
Chan Veasna	012 412 333
Pou Sambath	077 545 221

Done?  
Oh no, it is still not in 1NF yet.  
Remove Repeating row.

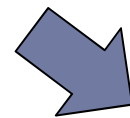
<u>StudyID</u>	Course Name	T.ID
1	Database	T1
2	Database	T2
3	Web Prog	T3
4	Web Prog	T3
5	Networking	T4



<u>Teacher Name</u>	Teacher Tel
Sok Piseth	012 123 456
Sao Kanha	0977 322 111
Chan Veasna	012 412 333
Pou Sambath	077 545 221

### Note about primary key:

- In theory, you can choose *TeacherName* to be a primary key.
- But in practice, you should add *TeacherID* as the primary key.



<u>ID</u>	Teacher Name	Teacher Tel
T1	Sok Piseth	012 123 456
T2	Sao Kanha	0977 322 111
T3	Chan Veasna	012 412 333
T4	Pou Sambath	077 545 221



## Boyce Codd Normal Form (BCNF) – 3.5NF

---

### The official qualifications for BCNF are:

1. A table is already in 3NF.
2. All determinants must be superkeys.

All determinants that are not superkeys are removed to place in another table.

$K$  is a *superkey* for relation  $R$  if  $K$  functionally determines all of  $R$ .

$K$  is a *(candidate)key* for  $R$  if  $K$  is a superkey, but no proper subset of  $K$  is a superkey.

---



## Boyce Codd Normal Form (BCNF) (Cont.)

- Example of a table not in BCNF:

<u>Student</u>	<u>Course</u>	Teacher
Sok	DB	John
Sao	DB	William
Chan	E-Commerce	Todd
Sok	E-Commerce	Todd
Chan	DB	William

- Key: {Student, Course}
- Functional Dependency:
  - {Student, Course} → Teacher
  - Teacher → Course
- Problem: **Teacher** is not a superkey but determines **Course**.

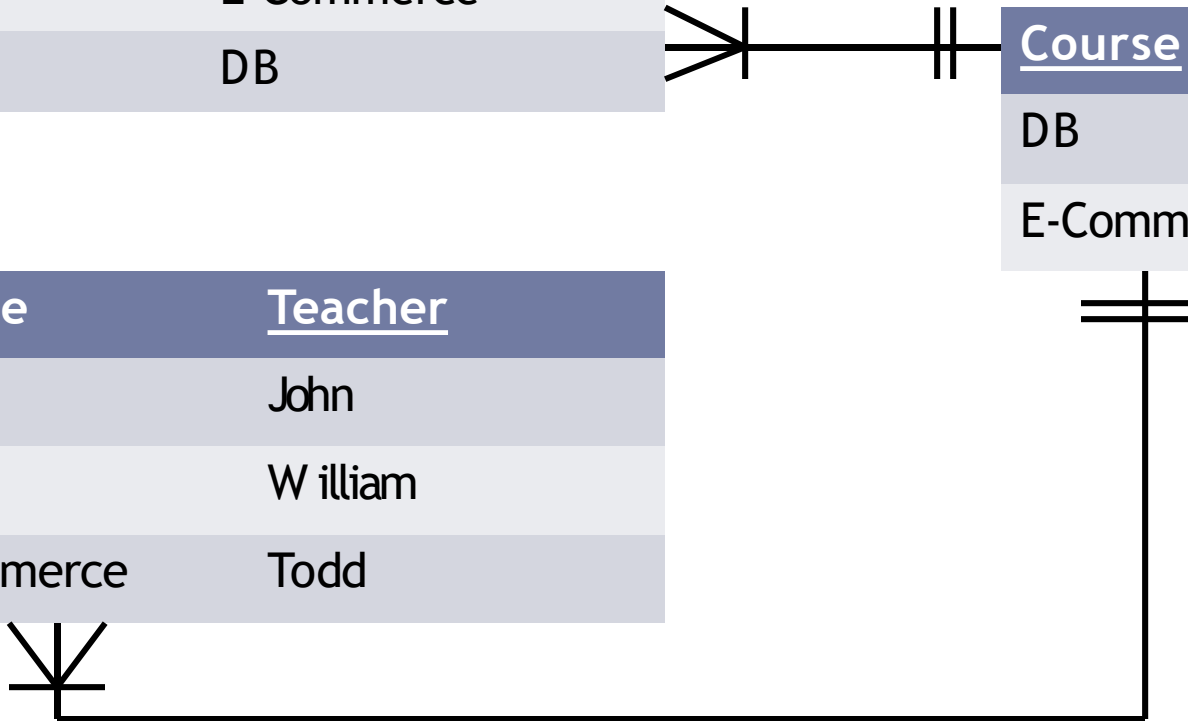


<u>Student</u>	<u>Course</u>
Sok	DB
Sao	DB
Chan	E-Commerce
Sok	E-Commerce
Chan	DB

**Solution:** Decouple a table contains **Teacher** and **Course** from original table (**Student**, **Course**). Finally, connect the new and old table to third table contains **Course**.

<u>Course</u>	<u>Teacher</u>
DB	John
DB	William
E-Commerce	Todd

<u>Course</u>
DB
E-Commerce



## Forth Normal Form (4NF)

### The official qualifications for 4NF are:

1. A table is already in BCNF.
  2. A table contains no multi-valued dependencies.
- Multi-valued dependency: MVDs occur when two or more independent multi valued facts about the same attribute occur within the same table.
- $A \twoheadrightarrow B$  (B multi-valued depends on A)



## Example: MVD

Customer(name, addr, phones, email\_drinksLiked)

A drinker's phones are independent of the drinks they like.

name->->phones and name->->drinksLiked.

Thus, each of a drinker's phones appears with each of the drinks they like in all combinations.



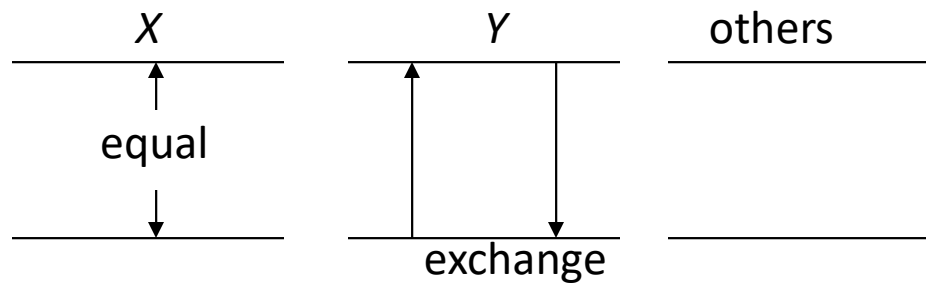
# Tuples Implied by **name**->->**phones**

If we have tuples:

name	addr	phones	drinksLiked
sue	a	p1	d1
sue	a	p2	d2
sue	a	p2	d1
sue	a	p1	d2

Then these tuples must also be in the relation.

# Picture of MVD $X \dashrightarrow Y$



## Forth Normal Form (4NF) (Cont.)

- Example of a table not in 4NF:

Student	Major	Hobby
Sok	IT	Football
Sok	IT	Volleyball
Sao	IT	Football
Sao	Med	Football
Chan	IT	NULL
Puth	NULL	Football
Tith	NULL	NULL

- Key: {Student, Major, Hobby}
- MVD: Student -->> Major, Hobby

**Solution:** Decouple to each table contains MVD. Finally, connect each to a third table contains *Student*.

<u>Student</u>
Sok
Sao
Chan
Puth
Tith

<u>Student</u>	<u>Major</u>
Sok	IT
Sao	IT
Sao	Med
Chan	IT
Puth	NULL
Tith	NULL

<u>Student</u>	<u>Hobby</u>
Sok	Football
Sok	Volleyball
Sao	Football
Chan	NULL
Puth	Football
Tith	NULL



## Fifth Normal Form (5NF)

The official qualifications for 5NF are:

1. A table is already in 4NF.
2. The attributes of multi-valued dependencies are related.

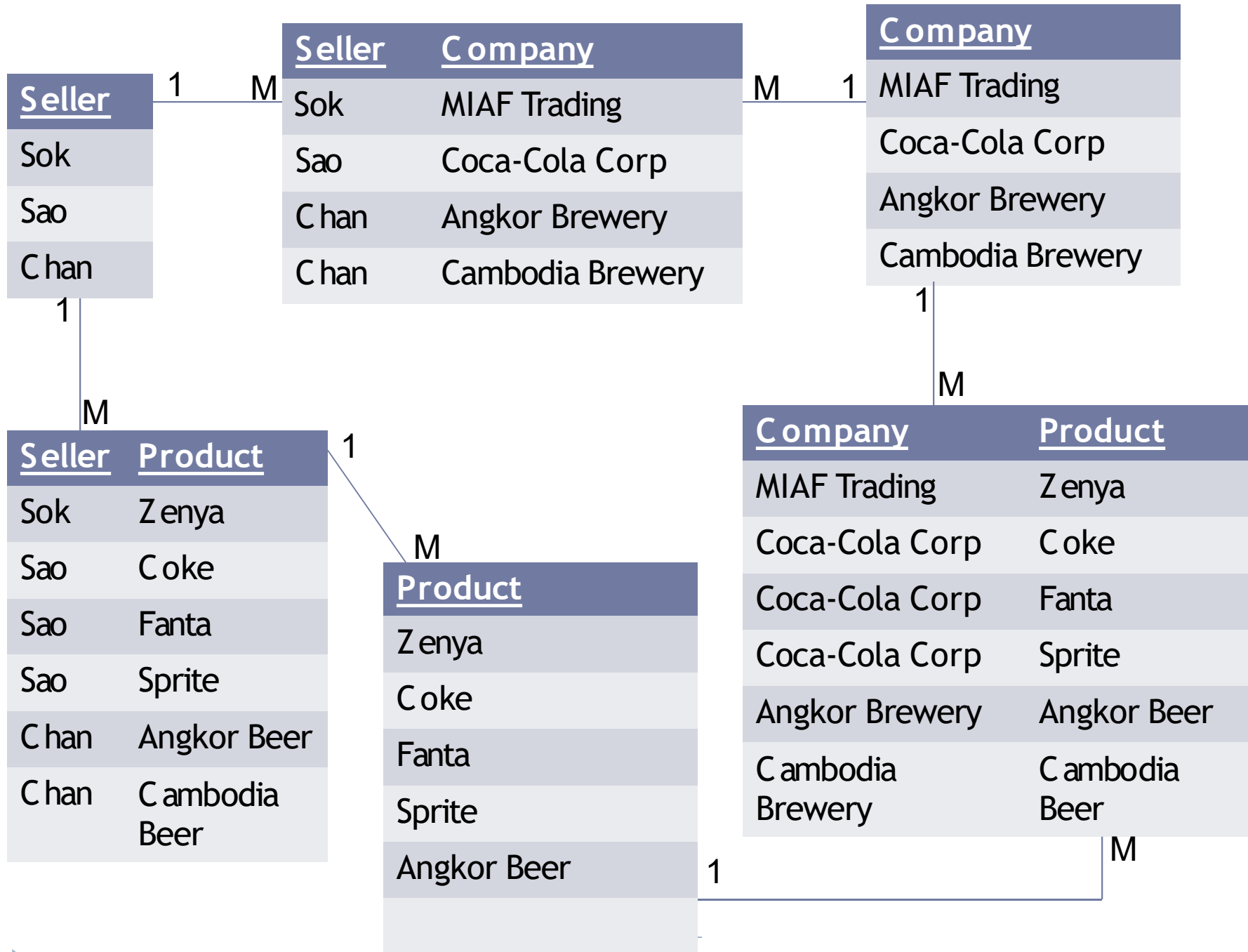


## Fifth Normal Form (5NF) (Cont.)

- Example of a table not in 5NF:

<u>Seller</u>	<u>Company</u>	<u>Product</u>
Sok	MIAF Trading	Zenya
Sao	Coca-Cola Corp	Coke
Sao	Coca-Cola Corp	Fanta
Sao	Coca-Cola Corp	Sprite
Chan	Angkor Brewery	Angkor Beer
Chan	Cambodia Brewery	Cambodia Beer

- Key: {Seller, Company, Product}
- MVD: Seller -->> Company, Product
- *Product* is related to *Company*.



# Normalization Practice

**Please see the normalization example slides**



# Acknowledgement

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