Functional Dependencies

UNIT III

Overview

- Functional dependencies
- Examples of FDs
- Inference rules

Functional Dependencies

- Functional dependencies (FDs)
  - Are used to specify *formal measures* of the "goodness" of relational designs
  - And keys are used to define **normal forms** for relations
  - Are **constraints** that are derived from the *meaning* and *interrelationships* of the data attributes
- A set of attributes X *functionally determines* a set of attributes Y if the value of X determines a unique value for Y

Functional Dependencies

- X -> Y holds if whenever two tuples have the same value for X, they *must have* the same value for Y
  - For any two tuples t1 and t2 in any relation instance r(R): If t1[X]=t2[X], *then* t1[Y]=t2[Y]
- X -> Y in R specifies a *constraint* on all relation instances r(R)
- Written as X -> Y; can be displayed graphically on a relation schema as in Figures. (denoted by the arrow: →).
- FDs are derived from the real-world constraints on the attributes

Examples of FD constraints

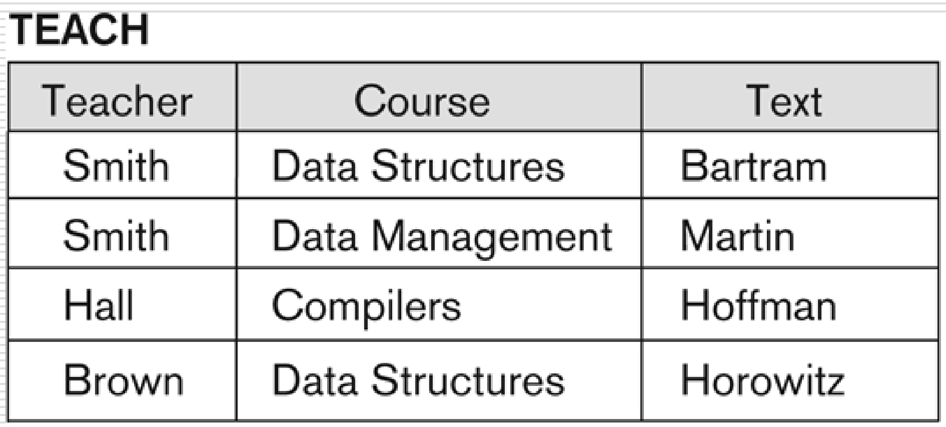
- Social security number determines employee name
  - SSN -> ENAME
- Project number determines project name and location
  - PNUMBER -> {PNAME, PLOCATION}
- Employee ssn and project number determines the hours per week that the employee works on the project
  - {SSN, PNUMBER} -> HOURS

Examples of FD constraints

- An FD is a property of the attributes in the schema R
- The constraint must hold on *every* relation instance r(R)
- If K is a key of R, then K functionally determines all attributes in R
  - (since we never have two distinct tuples with t1[K]=t2[K])

Examples of FD constraints

- FD's are a property of the meaning of data and hold at all times.
- Certain FD's can be ruled out based on a given state of the database



TEACH

Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz

Figure 10.7
A relation state of TEACH with a *possible* functional dependency TEXT → COURSE. However, TEACHER → COURSE is ruled out.

Inference Rules for FDs

- Given a set of FDs F, we can **infer** additional FDs that hold whenever the FDs in F hold
- Armstrong's inference rules:
  - IR1. (**Reflexive**) If Y *subset-of* X, then X -> Y
  - IR2. (**Augmentation**) If X -> Y, then XZ -> YZ
    - (Notation: XZ stands for X U Z)
  - IR3. (**Transitive**) If X -> Y and Y -> Z, then X -> Z
- IR1, IR2, IR3 form a **sound** and **complete** set of inference rules
  - These are rules hold and all other rules that hold can be deduced from these

Inference Rules for FDs

- Some additional inference rules that are useful:
  - **Decomposition:** If X -> YZ, then X -> Y and X -> Z
  - **Union:** If X -> Y and X -> Z, then X -> YZ
  - **Psuedotransitivity:** If X -> Y and WY -> Z, then WX -> Z

- The last three inference rules, as well as any other inference rules, can be deduced from IR1, IR2, and IR3 (completeness property)

Inference Rules for FDs

- **Closure** of a set F of FDs is the set F^+ of all FDs that can be inferred from F
- **Closure** of a set of attributes X with respect to F is the set X^+ of all attributes that are functionally determined by X
- X^+ can be calculated by repeatedly applying IR1, IR2, IR3 using the FDs in F

Equivalence of Sets of FDs

- Two sets of FDs F and G are **equivalent** if:
  - Every FD in F can be inferred from G, and
  - Every FD in G can be inferred from F
  - Hence, F and G are equivalent if $F^+ = G^+$
- Definition (**Covers**):
  - F **covers** G if every FD in G can be inferred from F
    - (i.e., if G^+ *subset-of* F^+)
- F and G are equivalent if F covers G and G covers F
- There is an algorithm for checking equivalence of sets of FDs

Minimal Sets of FDs

- A set of FDs is **minimal** if it satisfies the following conditions:
  1. Every dependency in F has a single attribute for its RHS.
  2. We cannot remove any dependency from F and have a set of dependencies that is equivalent to F.
  3. We cannot replace any dependency X -> A in F with a dependency Y -> A, where Y proper-subset-of X (Y subset-of X) and still have a set of dependencies that is equivalent to F.

Minimal Sets of FDs

- Every set of FDs has an equivalent minimal set
- There can be several equivalent minimal sets
- There is no simple algorithm for computing a minimal set of FDs that is equivalent to a set F of FDs

Computing the Minimal Sets of FDs

We illustrate the above algorithm with the following:

Let the given set of FDs be *E* : {$B \rightarrow A$, $D \rightarrow A$, $AB \rightarrow D$}.We have to find the minimum
cover of *E*.

■ All above dependencies are in canonical form; so we have completed step 1
of Algorithm 10.2 and can proceed to step 2. In step 2 we need to determine
if $AB \rightarrow D$ has any redundant attribute on the left-hand side; that is, can it be
replaced by $B \rightarrow D$ or $A \rightarrow D$?

■ Since B → A, by augmenting with *B* on both sides (IR2), we have $BB \rightarrow AB$, or
$B \rightarrow AB$ (i). However, $AB \rightarrow D$ as given (ii).

■ Hence by the transitive rule (IR3), we get from (i) and (ii), $B \rightarrow D$. Hence
$AB \rightarrow D$ may be replaced by $B \rightarrow D$.

■ We now have a set equivalent to original *E* , say *E′* : {$B \rightarrow A$, $D \rightarrow A$, $B \rightarrow D$}.
No further reduction is possible in step 2 since all FDs have a single attribute
on the left-hand side.

■ In step 3 we look for a redundant FD in E′. By using the transitive rule on
$B \rightarrow D$ and $D \rightarrow A$, we derive $B \rightarrow A$. Hence $B \rightarrow A$ is redundant in E′ and can
be eliminated.

■ Hence the minimum cover of E is {$B \rightarrow D$, $D \rightarrow A$}.

Normal Forms Based on Primary Keys

- Normalization of Relations
- Practical Use of Normal Forms
- Definitions of Keys and Attributes Participating in Keys
- First Normal Form
- Second Normal Form
- Third Normal Form

Normalization of Relations

- Proposed by Codd(1972) – three NFs
- Boyce and Codd proposed BCNF
- **Normalization:**
  - The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations
- **Normal form:**
  - Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form

Normalization of Relations

- 2NF, 3NF, BCNF
  - based on keys and FDs of a relation schema
- 4NF
  - based on keys, multi-valued dependencies : MVDs;
- 5NF
  - based on keys, join dependencies : JDs
- Additional properties may be needed to ensure a good relational design
  - lossless join
  - dependency preservation

Practical Use of Normal Forms

- **Normalization** is carried out in practice so that the resulting designs are of high quality and meet the desirable properties
- The practical utility of these normal forms becomes questionable when the constraints on which they are based are *hard to understand* or to *detect*
- The database designers *need not* normalize to the highest possible normal form
  - usually up to 3NF, BCNF or 4NF
- **Denormalization**:
  - The process of storing the join of higher normal form relations as a base relation—which is in a lower normal form

Definitions of Keys and Attributes in Keys

- A **superkey** of a relation schema R = {A1, A2, ..., An} is a set of attributes S *subset-of* R with the property that no two tuples t1 and t2 in any legal relation state r of R will have t1[S] = t2[S]

- A **key** K is a **superkey** with the *additional property* that removal of any attribute from K will cause K not to be a superkey any more.

Definitions of Keys and Attributes in Keys

- If a relation schema has more than one key, each is called a **candidate** key.
  - One of the candidate keys is *arbitrarily* designated to be the **primary key**, and the others are called **secondary keys**.
- A **Prime attribute** must be a member of *some* candidate key
- A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.

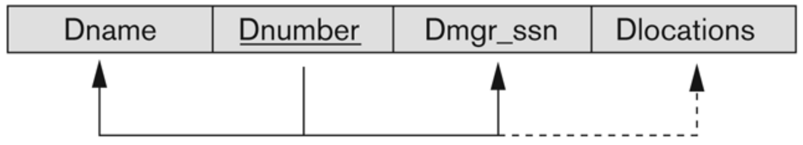
First Normal Form

- Disallows
  - composite attributes
  - multivalued attributes
  - **nested relations**; attributes whose values for an *individual tuple* are non-atomic
- Considered to be part of the definition of relation

Normalization into 1NF

(a)

DEPARTMENT



(b)

DEPARTMENT

Dname	<u>Dnumber</u>	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

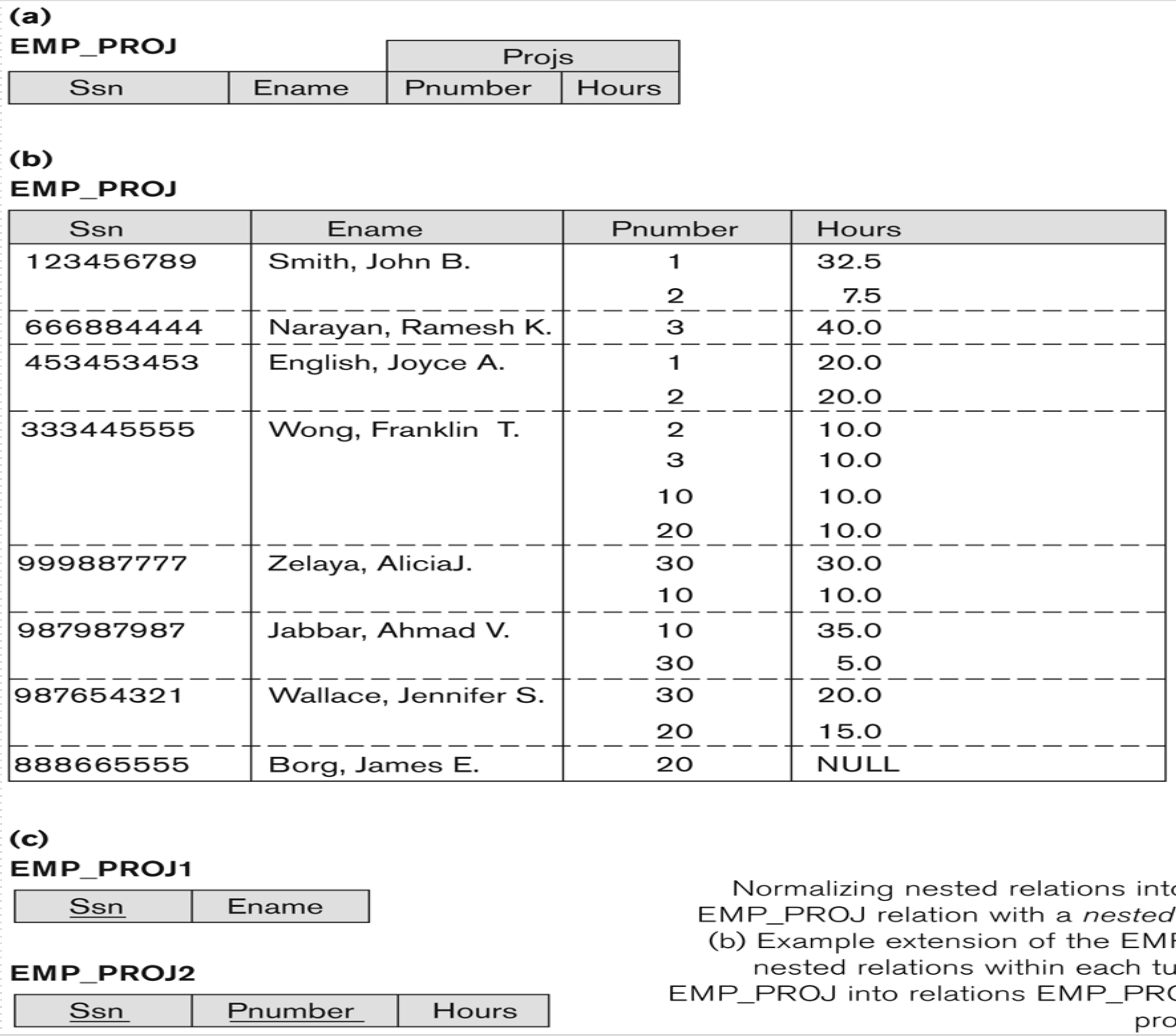
(c)

DEPARTMENT

Dname	<u>Dnumber</u>	Dmgr_ssn	<u>Dlocation</u>
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Figure 10.8
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Example state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

Normalization of nested relations into 1NF



(a)

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

(b)

EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, AliciaJ.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

(c)

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

Figure 10.9
Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a *nested relation* attribute PROJS. (b) Example extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

Second Normal Form

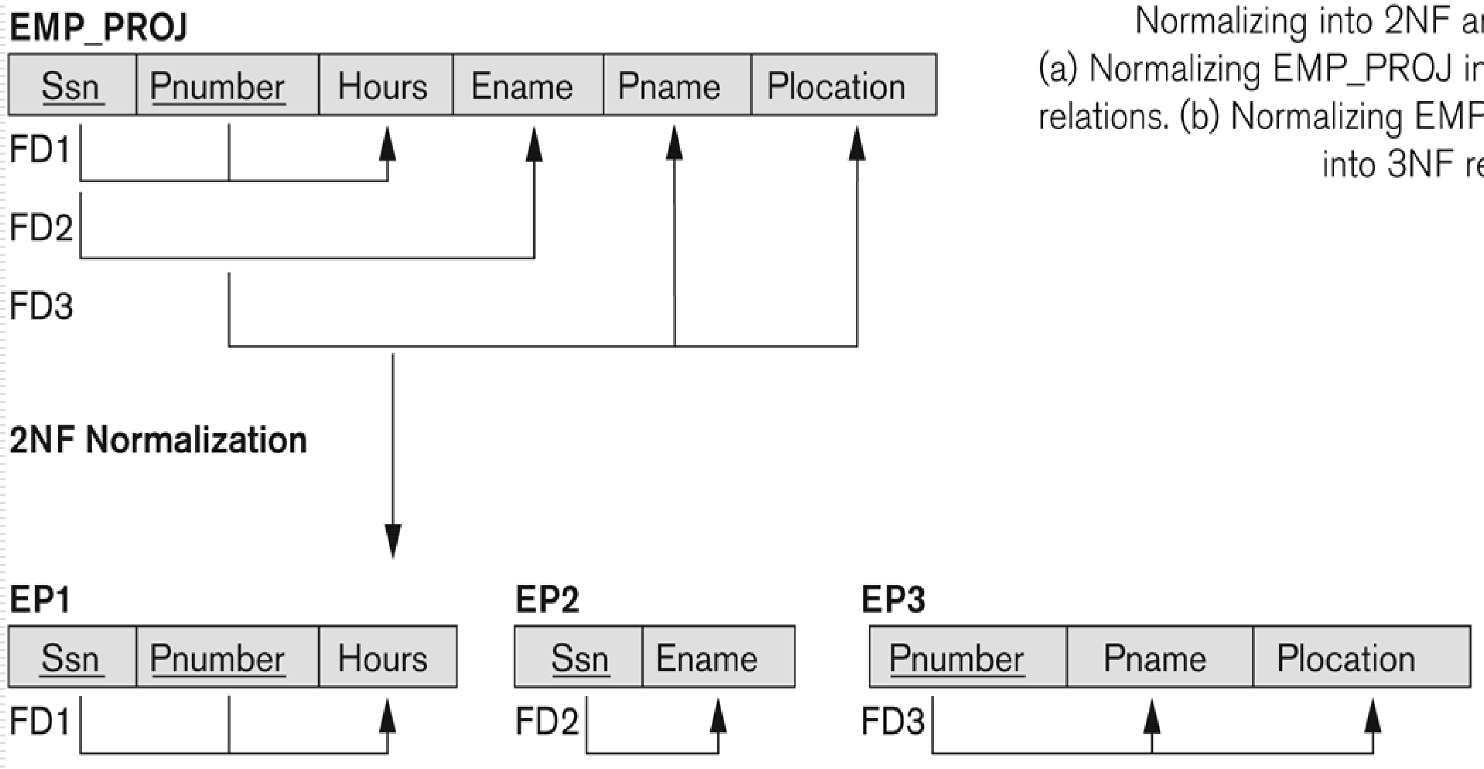
- Uses the concepts of **FDs, primary key**
- Definitions
  - **Prime attribute:** An attribute that is member of the primary key K
  - **Full functional dependency:** a FD Y -> Z where removal of any attribute from Y means the FD does not hold any more
- Examples:
  - {SSN, PNUMBER} -> HOURS is a full FD since neither SSN -> HOURS nor PNUMBER -> HOURS hold
  - {SSN, PNUMBER} -> ENAME is not a full FD (it is called a partial dependency) since SSN -> ENAME also holds

Second Normal Form

- A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on the primary key

- R can be decomposed into 2NF relations via the process of 2NF normalization

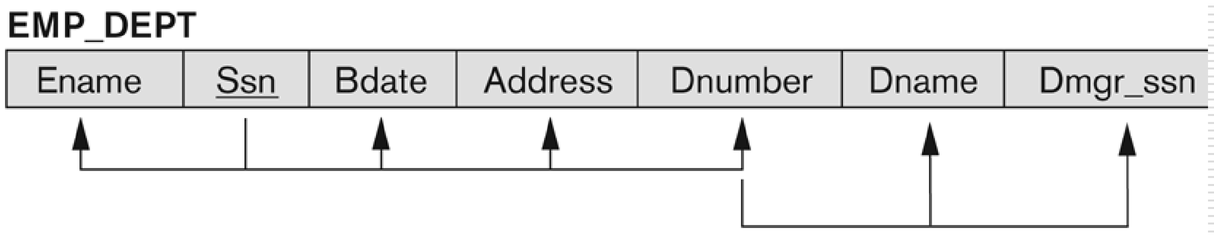
Normalizing into 2NF



Normalizing into 2NF and 3NF. (a) Normalizing EMP_PROJ into 2NF relations. (b) Normalizing EMP_DEPT into 3NF relations.

Third Normal Form

- Definition:
  - **Transitive functional dependency:** a FD X -> Z that can be derived from two FDs X -> Y and Y -> Z

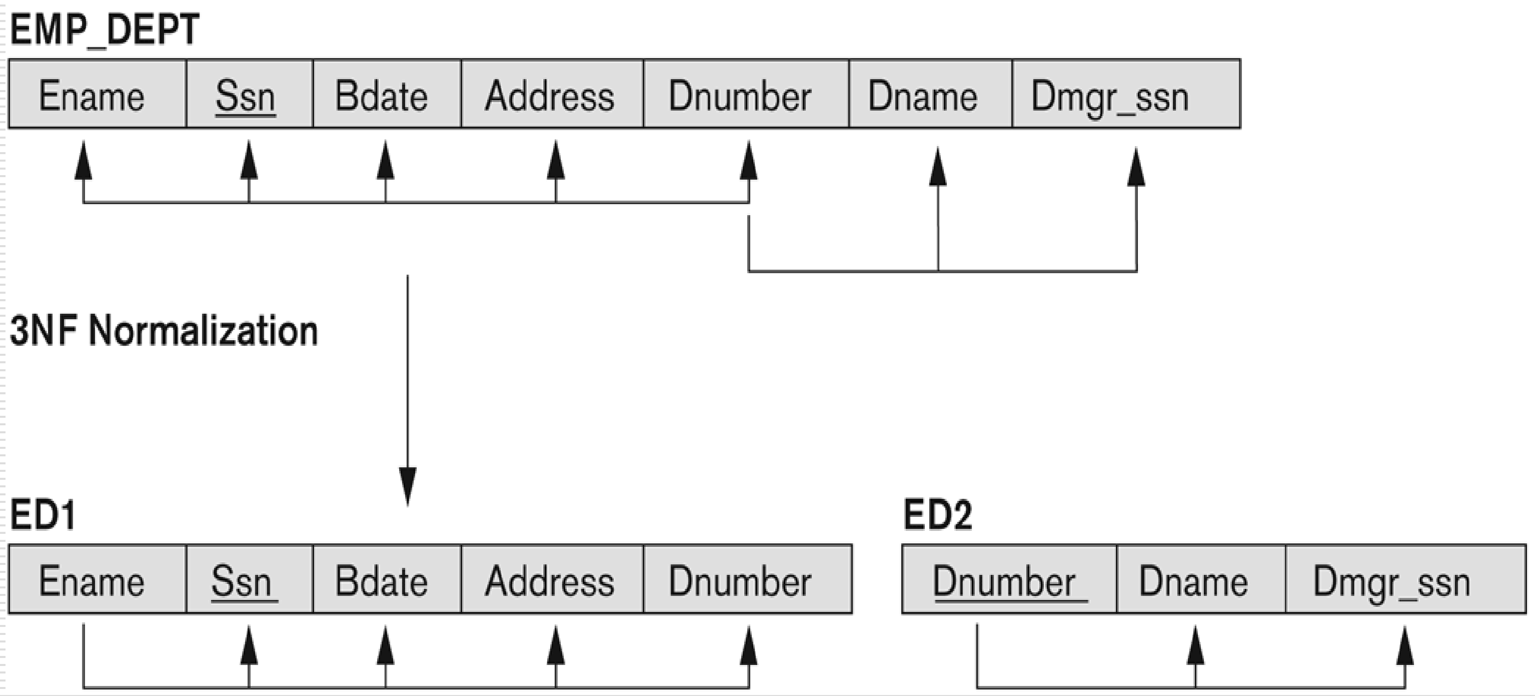


- Examples:
  - SSN -> DMGRSSN is a **transitive** FD
    - Since SSN -> DNUMBER and DNUMBER -> DMGRSSN hold
  - SSN -> ENAME is **non-transitive**
    - Since there is no set of attributes X where SSN -> X and X -> ENAME

Third Normal Form

- A relation schema R is in **third normal form (3NF)** if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on the primary key
- R can be decomposed into 3NF relations via the process of 3NF normalization
- NOTE:
  - In X -> Y and Y -> Z, with X as the primary key, we consider this a problem only if Y is not a candidate key.
  - When Y is a candidate key, there is no problem with the transitive dependency .
  - E.g., Consider EMP (SSN, Emp#, Salary).
    - Here, SSN -> Emp# -> Salary and Emp# is a candidate key.

Normalizing into 3NF



Normal Forms Defined Informally

- 1st normal form
  - All attributes depend on **the key**
- 2nd normal form
  - All attributes depend on **the whole key**
- 3rd normal form
  - All attributes depend on **nothing but the key**

Successive Normalization of LOTS into 2NF and 3NF

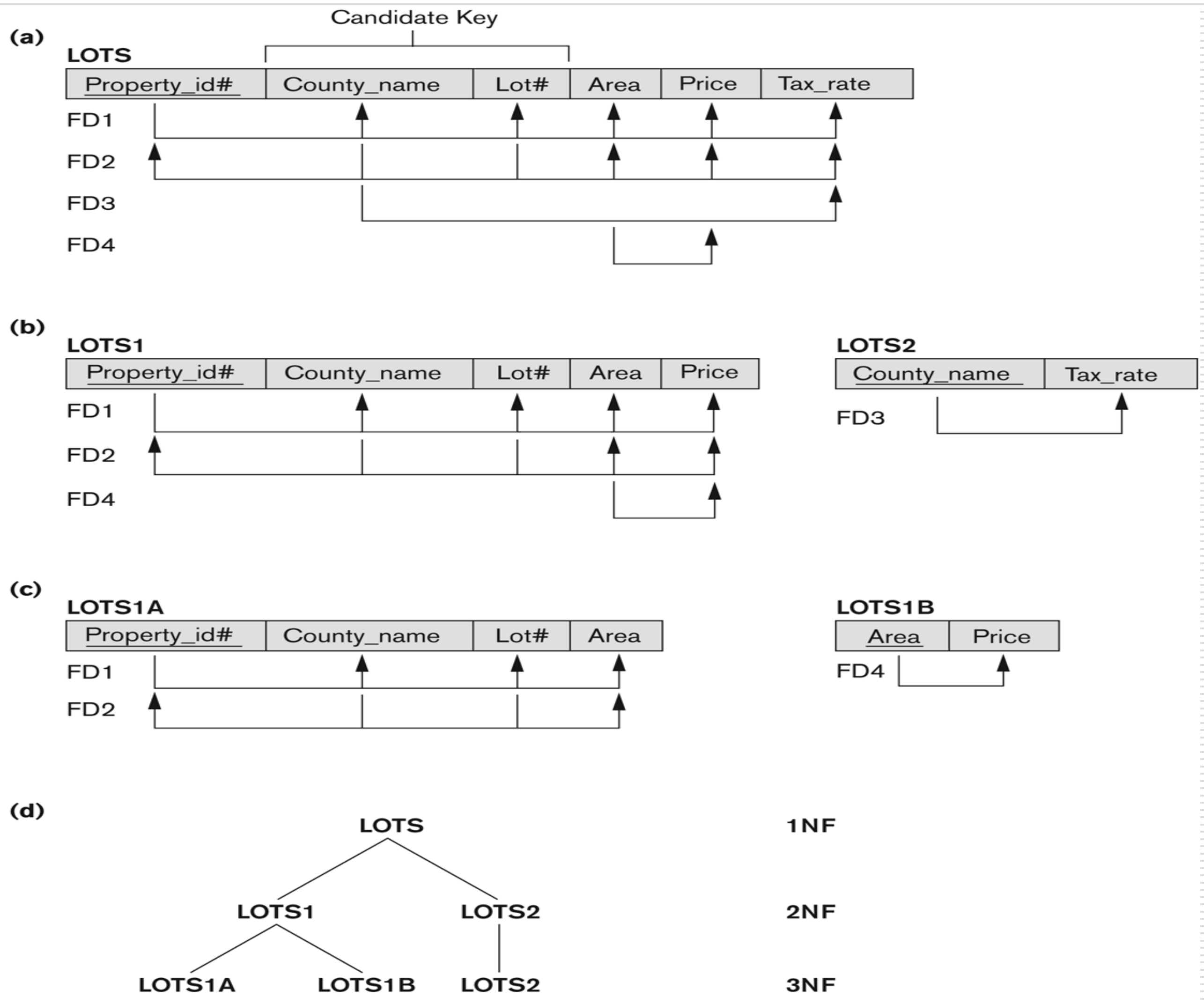


Figure 10.11
Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Summary of the progressive normalization of LOTS.

SUMMARY OF NORMAL FORMS based on Primary Keys

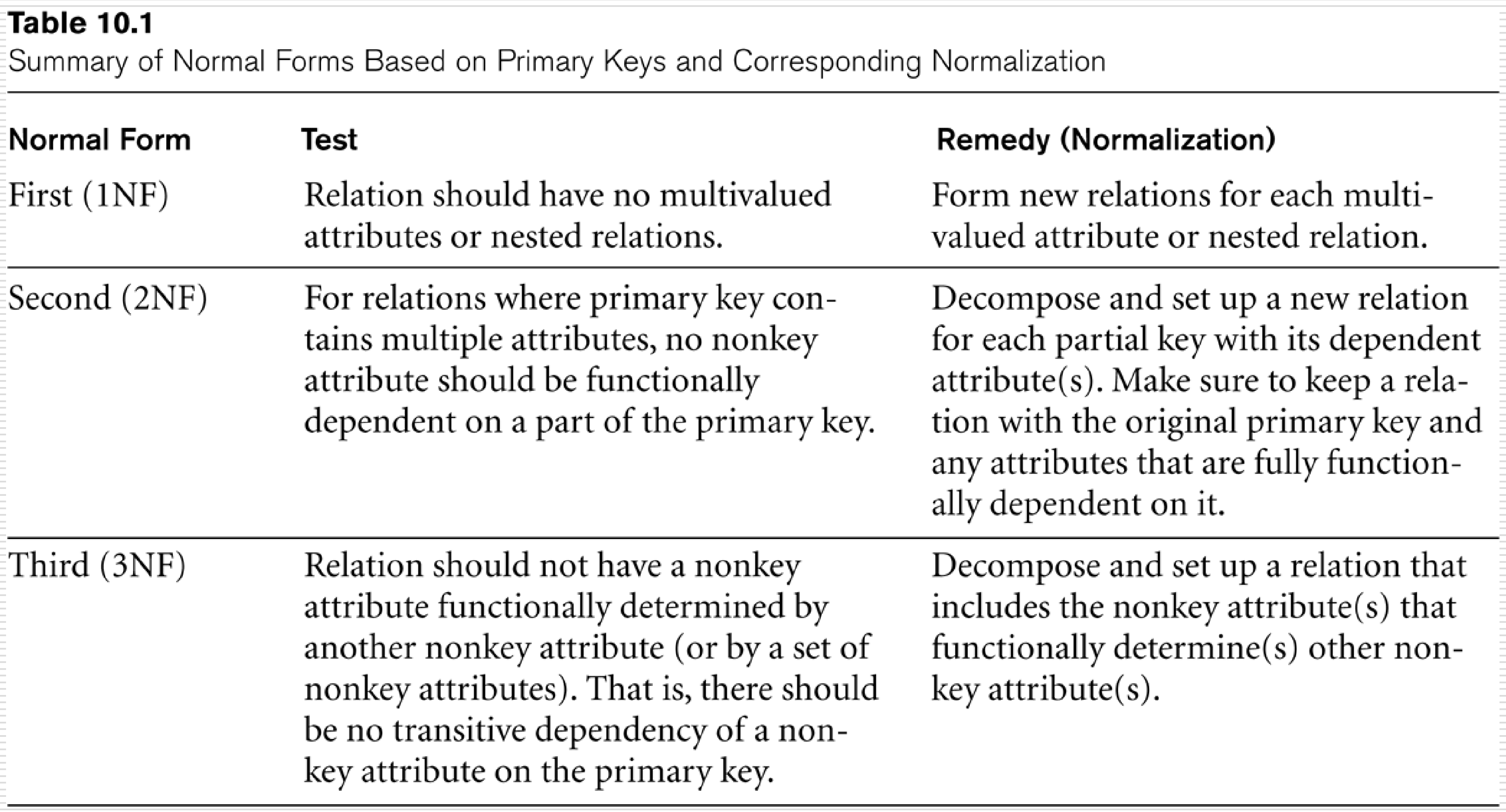


Table 10.1
Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

General Normal Form Definitions (For Multiple Keys)

- The above definitions consider the primary key only
- The following more general definitions take into account relations with multiple candidate keys
- A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on *every* key of R

General Normal Form Definitions

- Definition:
  - **Superkey** of relation schema R - a set of attributes S of R that contains a key of R
  - A relation schema R is in **third normal form (3NF)** if whenever a FD X -> A holds in R, then either:
    - (a) X is a superkey of R, or
    - (b) A is a prime attribute of R
- NOTE: Boyce-Codd normal form disallows condition (b) above

Example...

BCNF (Boyce-Codd Normal Form)

- A relation schema R is in **Boyce-Codd Normal Form (BCNF)** if whenever an **FD X -> A** holds in R, then **X is a superkey** of R
- Each normal form is strictly stronger than the previous one
  - Every 2NF relation is in 1NF
  - Every 3NF relation is in 2NF
  - Every BCNF relation is in 3NF
- There exist relations that are in 3NF but not in BCNF
- The goal is to have each relation in BCNF (or 3NF)

Boyce-Codd Normal Form

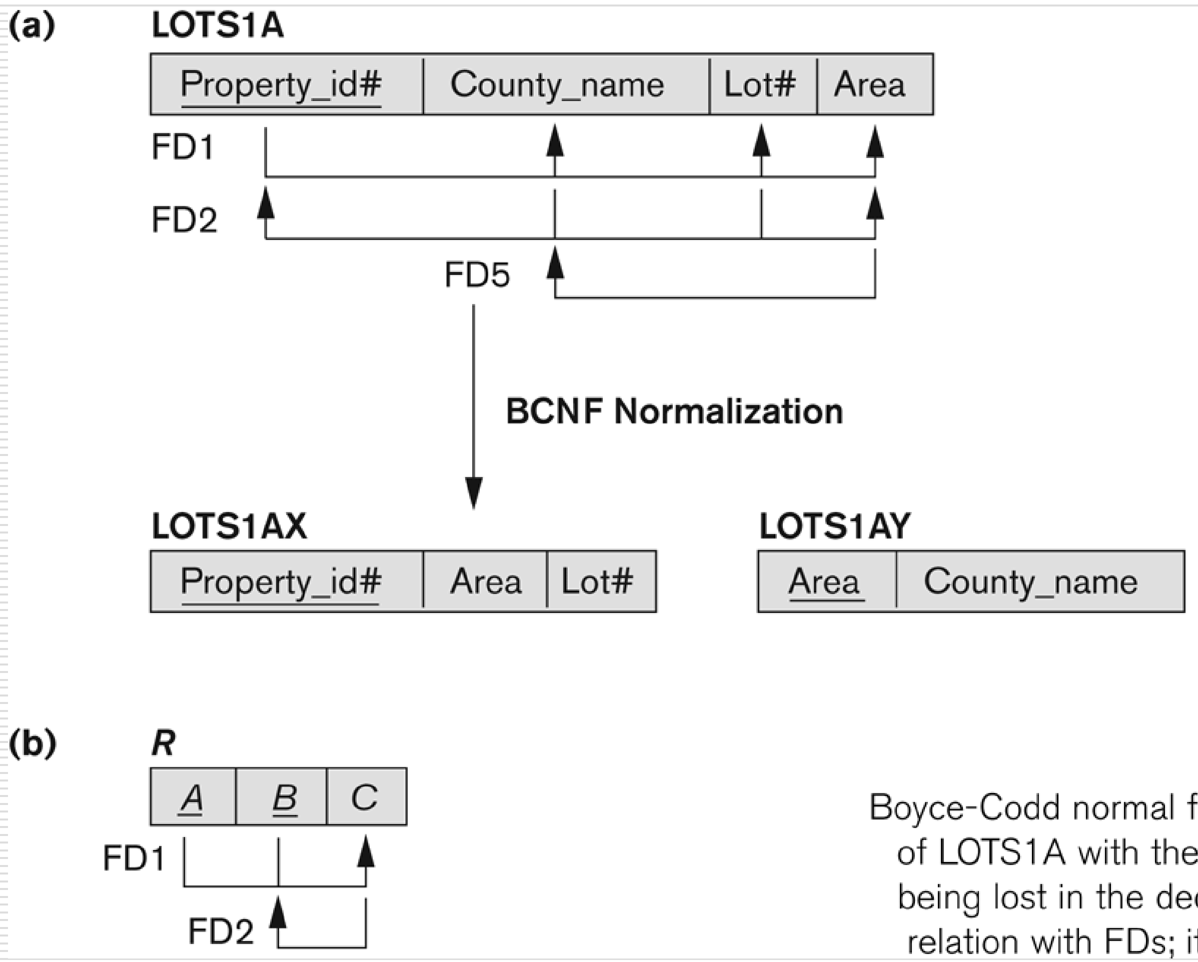
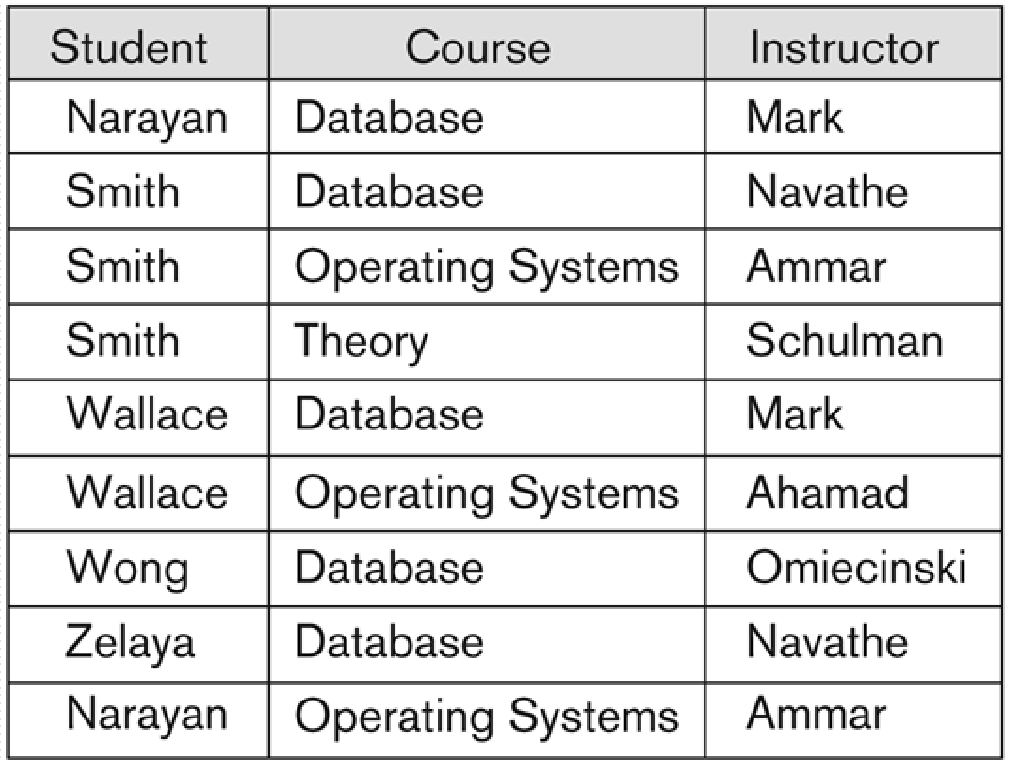


Figure 10.12
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

A relation TEACH that is in 3NF but not in BCNF



TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

Figure 10.13
A relation TEACH that is in 3NF but not BCNF.

Achieving the BCNF by Decomposition

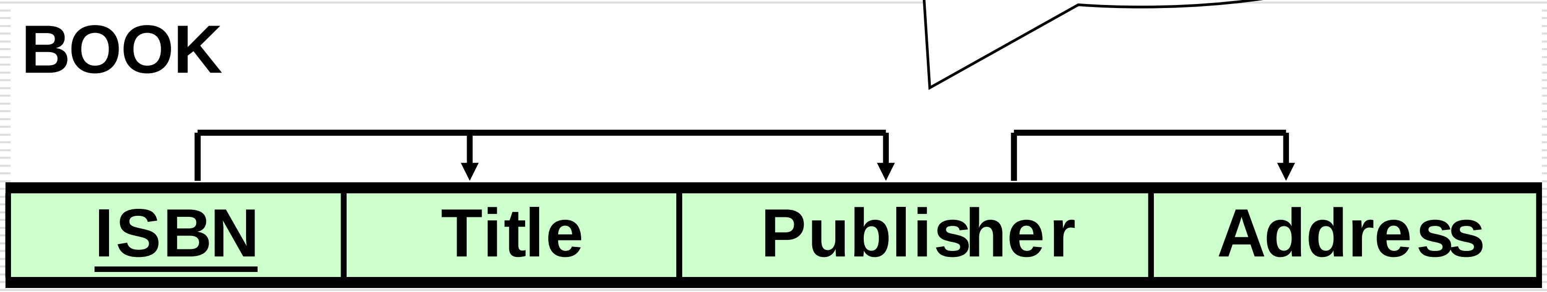
- Two FDs exist in the relation TEACH:
  - fd1: { student, course} -> instructor
  - fd2: instructor -> course
- {student, course} is a candidate key for this relation
  - So this relation is in 3NF *but not in* BCNF
- A relation **NOT** in BCNF should be decomposed so as to meet this property, while possibly forgoing the preservation of all functional dependencies in the decomposed relations.

Achieving the BCNF by Decomposition

- Three possible decompositions for relation TEACH
  - {student, instructor} and {student, course}
  - {course, instructor } and {course, student}
  - {instructor, course } and {instructor, student}
- All three decompositions will lose fd1.
  - We have to settle for sacrificing the functional dependency preservation. But we cannot sacrifice the non-additivity property after decomposition.
- Out of the above three, only the 3rd decomposition will not generate spurious tuples after join.(and hence has the non-additivity property).

Example 1: Determine NF

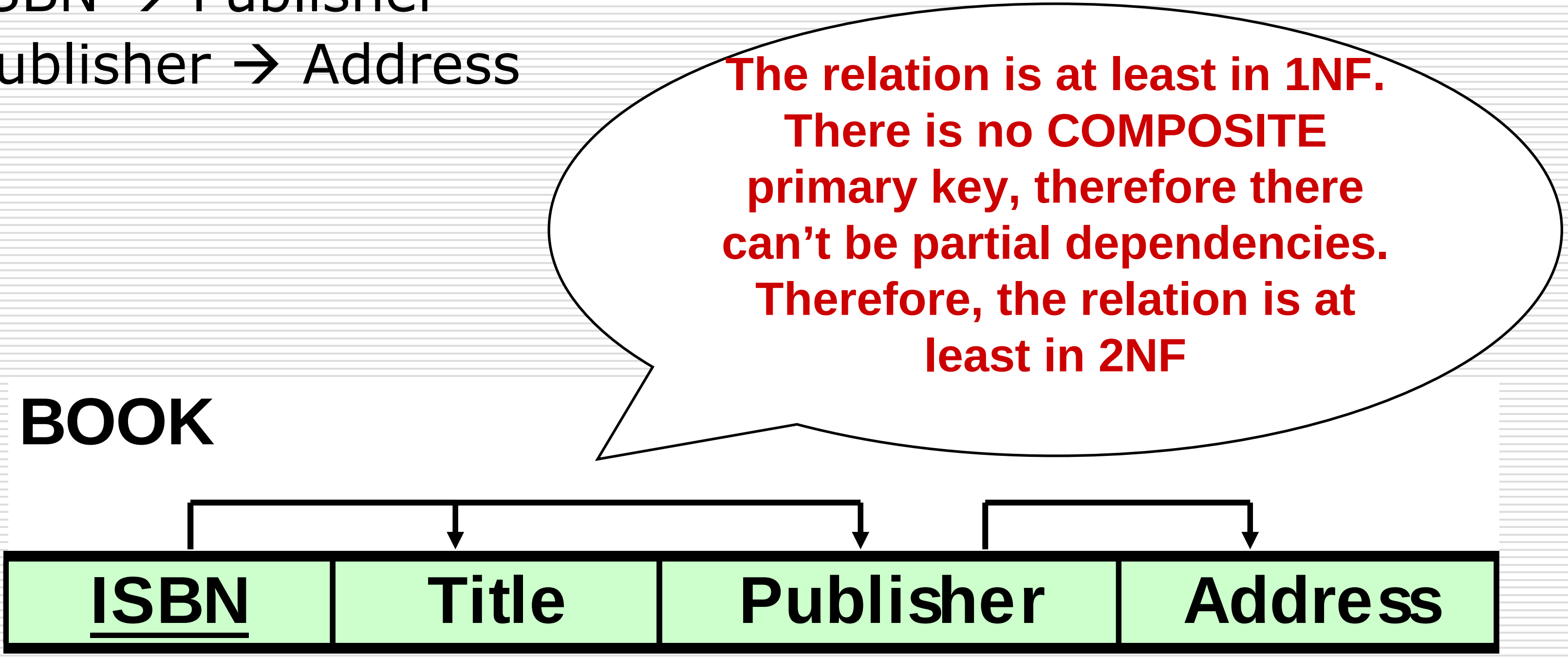
- ISBN → Title
- ISBN → Publisher
- Publisher → Address



All attributes are directly or indirectly determined by the primary key; therefore, the relation is at least in 1 NF

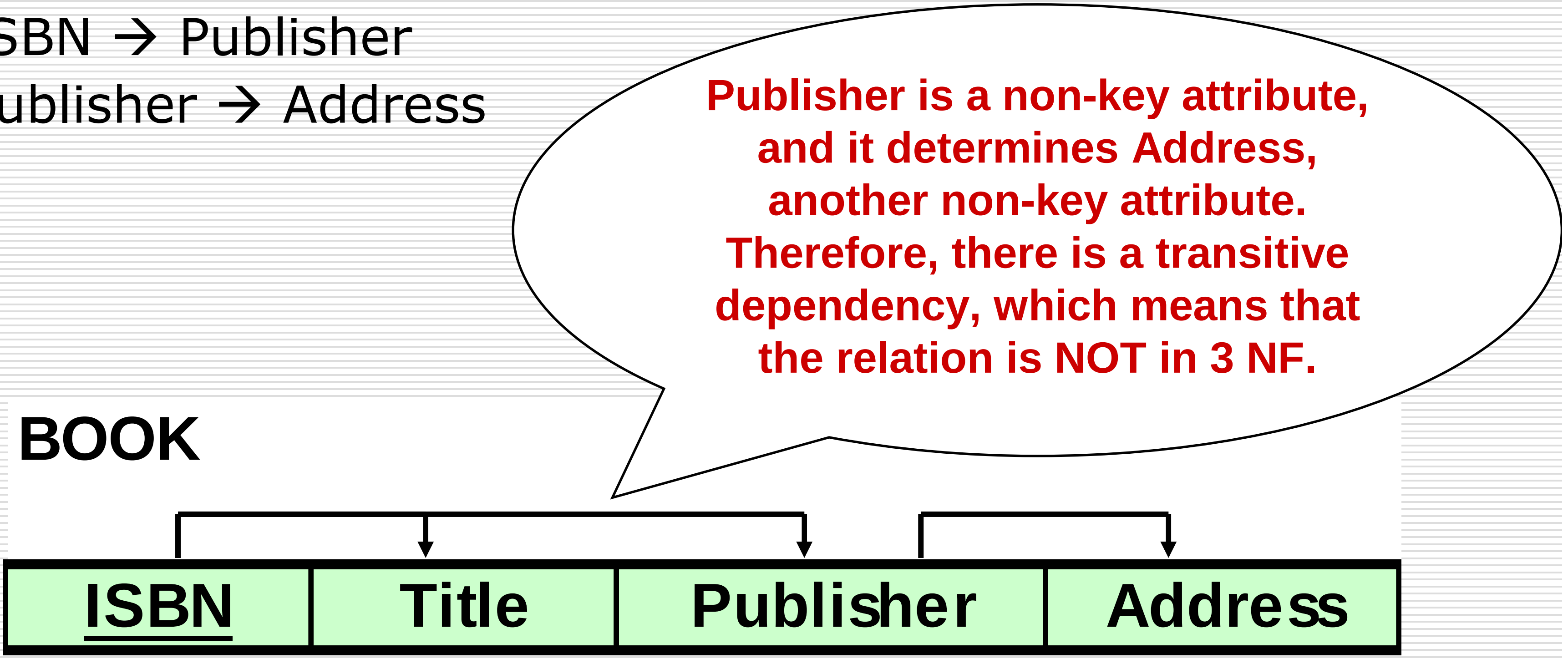
Example 1: Determine NF

- ISBN → Title
- ISBN → Publisher
- Publisher → Address


The relation is at least in 1NF. There is no COMPOSITE primary key, therefore there can't be partial dependencies. Therefore, the relation is at least in 2NF
BOOK
ISBN
Title
Publisher
Address

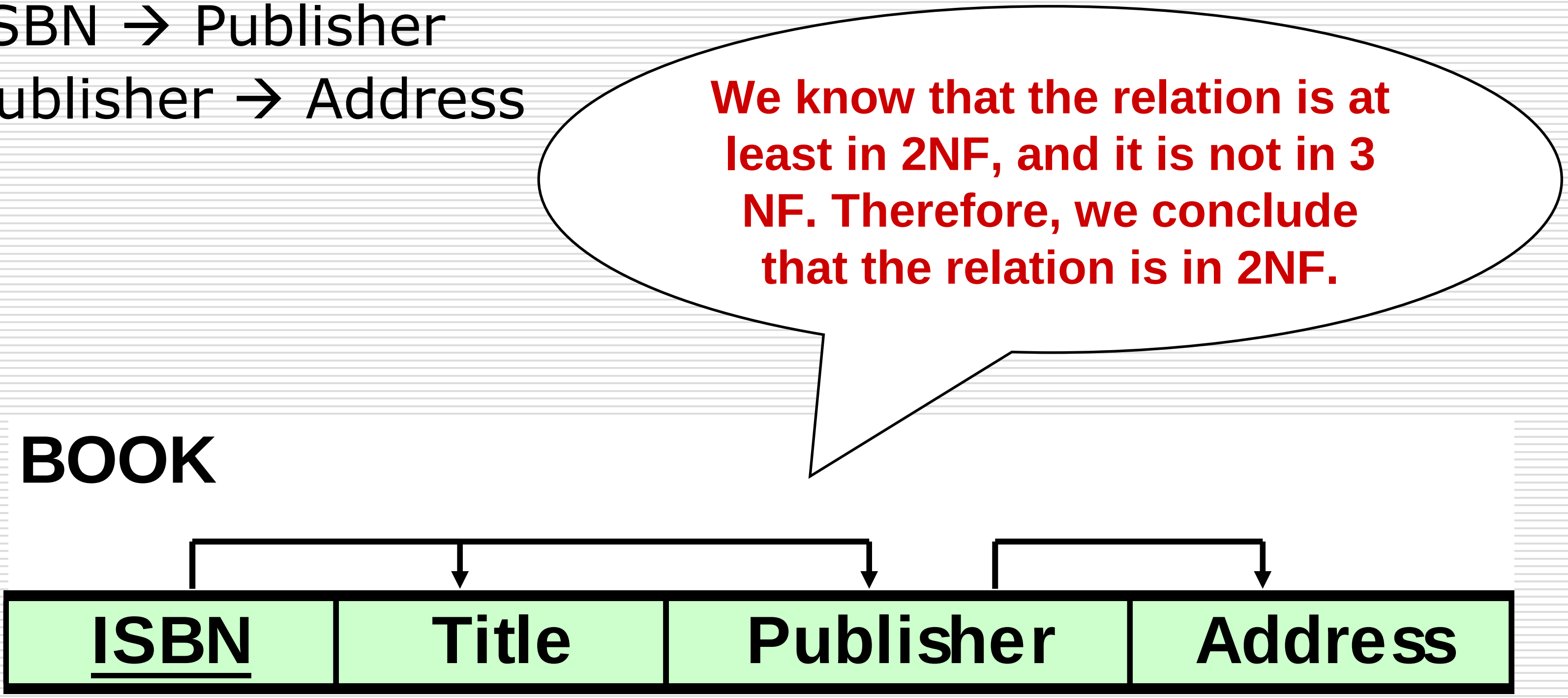
Example 1: Determine NF

- ISBN → Title
- ISBN → Publisher
- Publisher → Address


Publisher is a non-key attribute, and it determines Address, another non-key attribute. Therefore, there is a transitive dependency, which means that the relation is NOT in 3 NF.
BOOK
ISBN
Title
Publisher
Address

Example 1: Determine NF

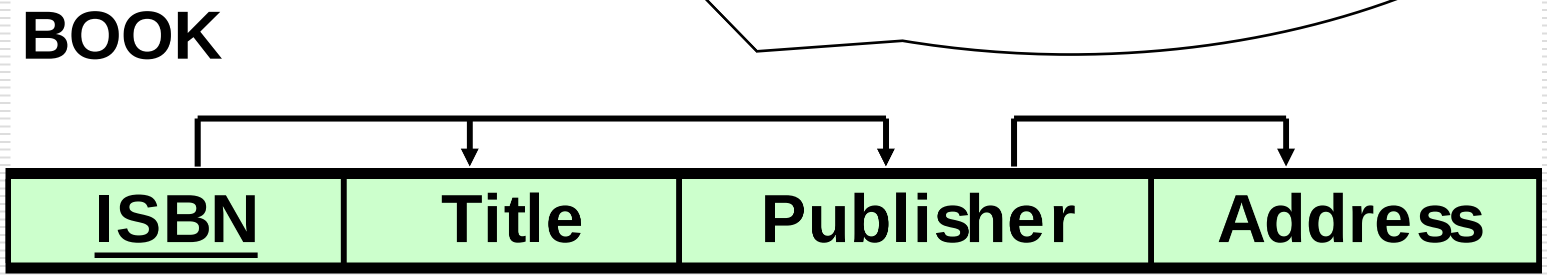
- ISBN → Title
- ISBN → Publisher
- Publisher → Address


We know that the relation is at least in 2NF, and it is not in 3 NF. Therefore, we conclude that the relation is in 2NF.
BOOK
ISBN
Title
Publisher
Address

Example 1: Determine NF

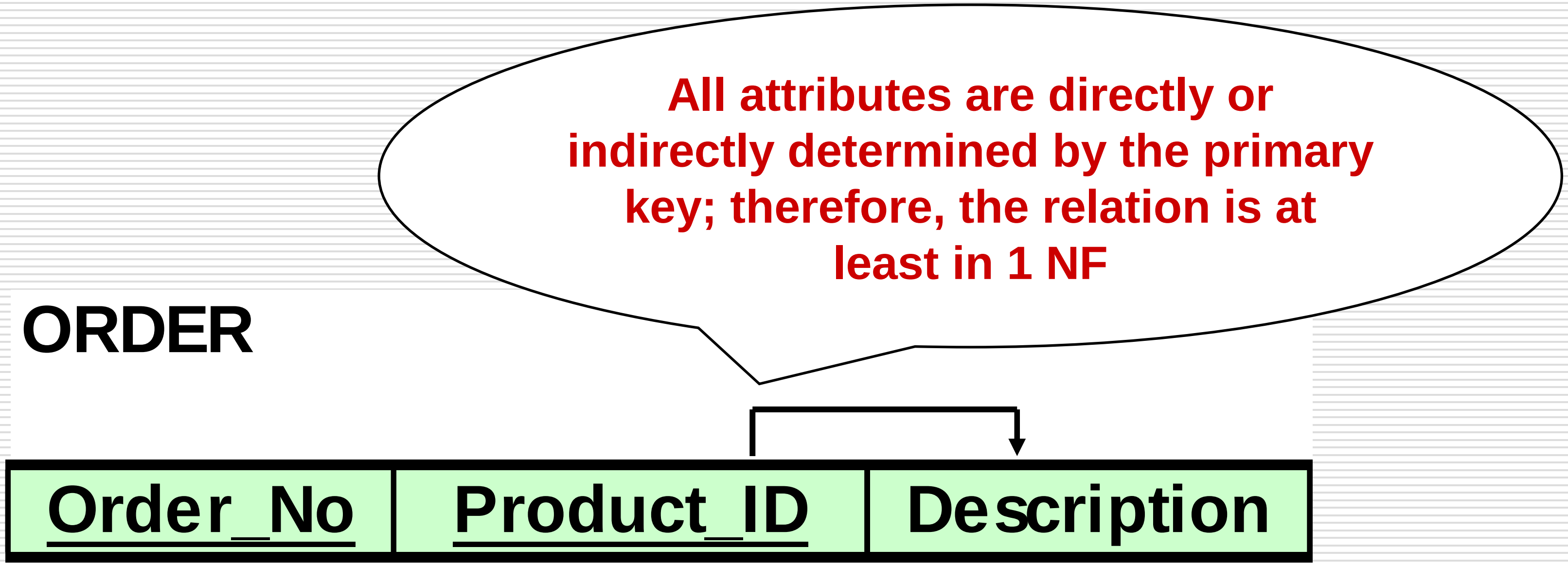
- ISBN → Title
- ISBN → Publisher
- Publisher → Address

In your solution you will write the following justification:
No M/V attributes, therefore at least 1NF
No partial dependencies, therefore at least 2NF
There is a transitive dependency (Publisher → Address), therefore, not 3NF
Conclusion: The relation is in 2NF



Example 2: Determine NF

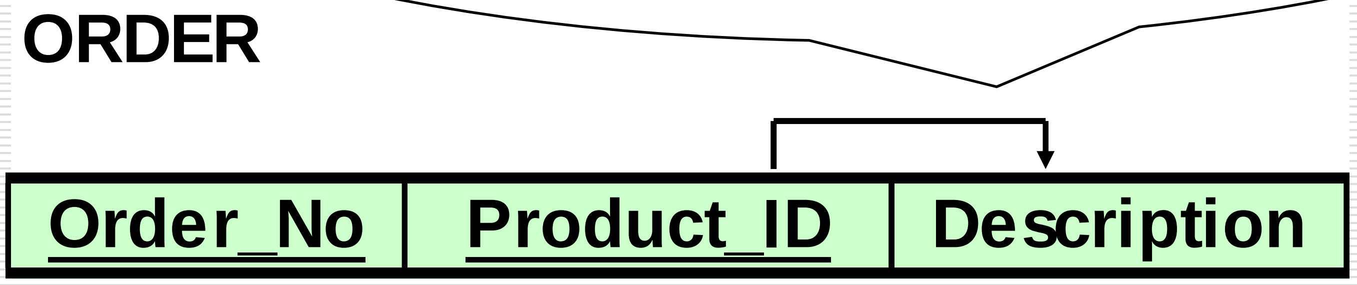
- Product_ID → Description


All attributes are directly or indirectly determined by the primary key; therefore, the relation is at least in 1 NF
ORDER
Order_No
Product_ID
Description

Example 2: Determine NF

- Product_ID → Description

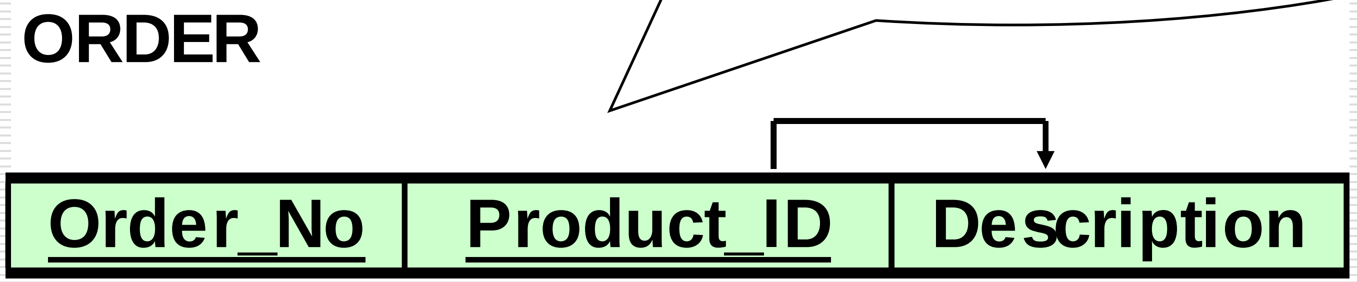
**The relation is at least in 1NF.
There is a COMPOSITE Primary Key (PK) (Order_No, Product_ID), therefore there can be partial dependencies. Product_ID, which is a part of PK, determines Description; hence, there is a partial dependency. Therefore, the relation is not 2NF. No sense to check for transitive dependencies!**



Example 2: Determine NF

- Product_ID → Description

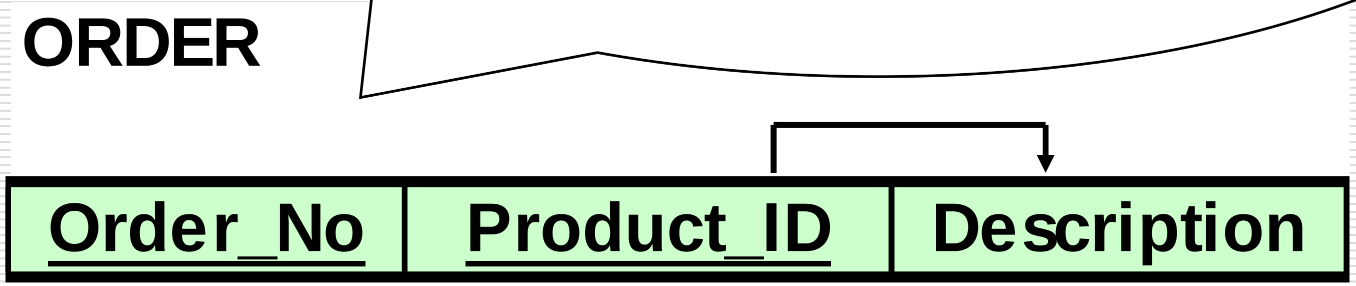
We know that the relation is at least in 1NF, and it is not in 2 NF. Therefore, we conclude that the relation is in 1 NF.



Example 2: Determine NF

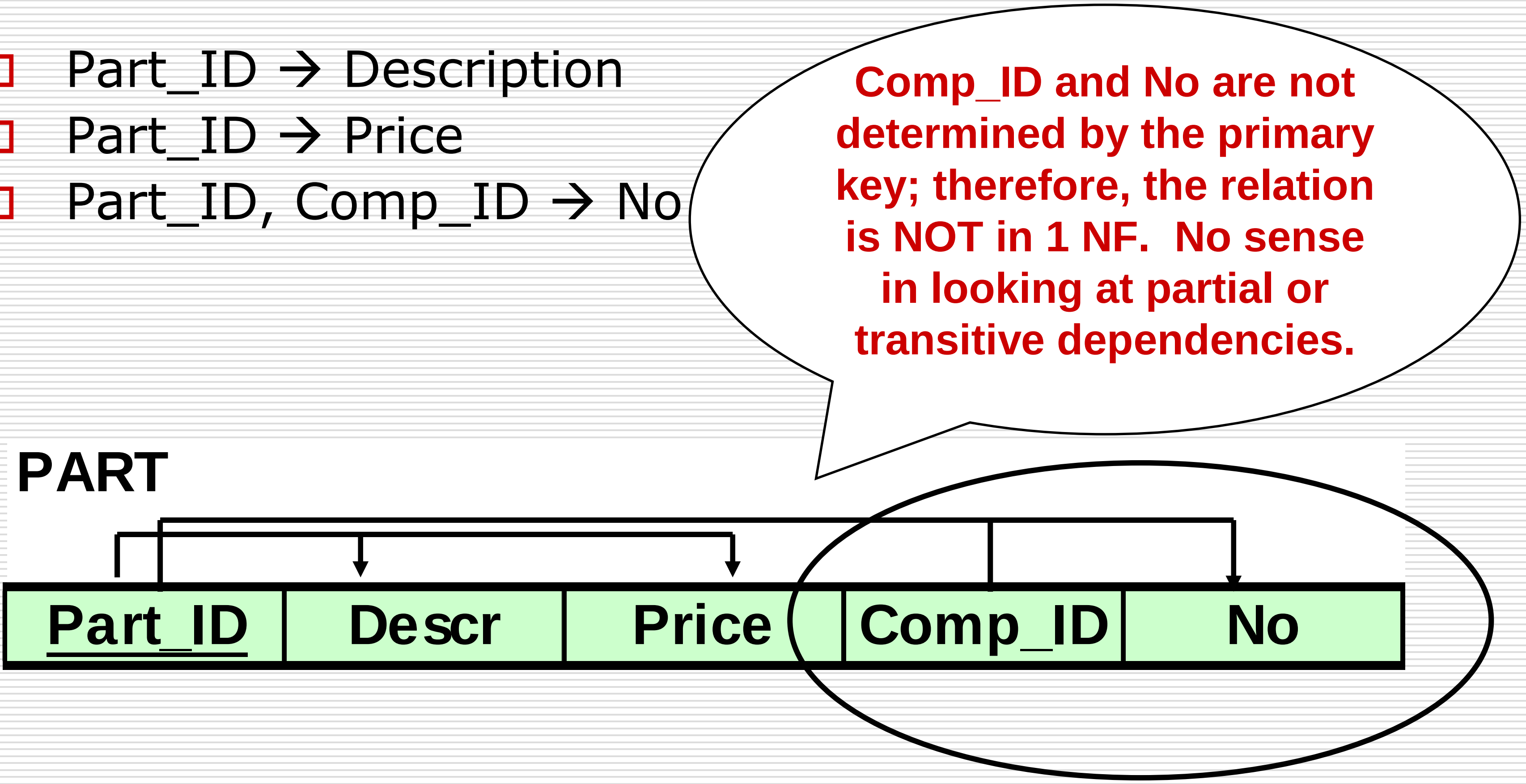
- Product_ID → Description

In your solution you will write the following justification:
1) No M/V attributes, therefore at least 1NF
2) There is a partial dependency (Product_ID → Description), therefore not in 2NF
Conclusion: The relation is in 1NF



Example 3: Determine NF

- Part_ID → Description
- Part_ID → Price
- Part_ID, Comp_ID → No

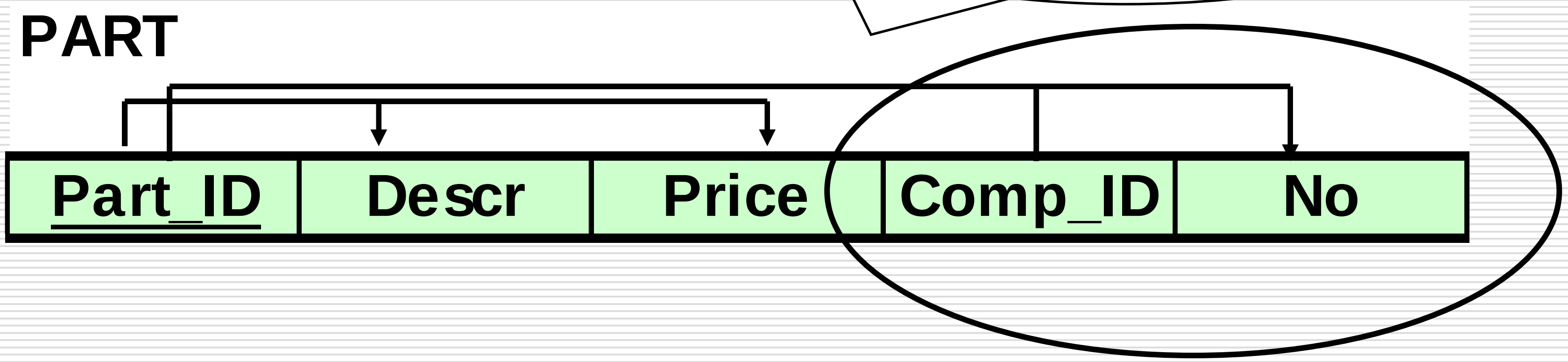


Comp_ID and No are not determined by the primary key; therefore, the relation is NOT in 1 NF. No sense in looking at partial or transitive dependencies.
PART
Part_ID
Descr
Price
Comp_ID
No

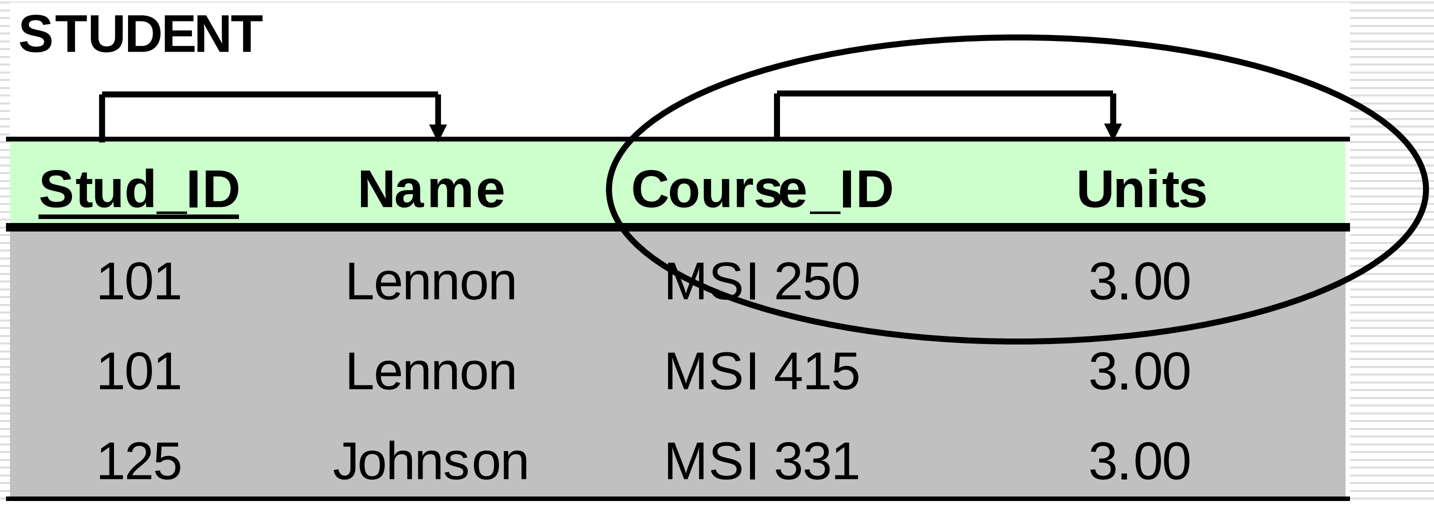
Example 3: Determine NF

- Part_ID → Description
- Part_ID → Price
- Part_ID, Comp_ID → No

In your solution you will write the following justification: There are M/V attributes; therefore, not 1NF
Conclusion: The relation is not normalized.



Bringing a Relation to 1NF

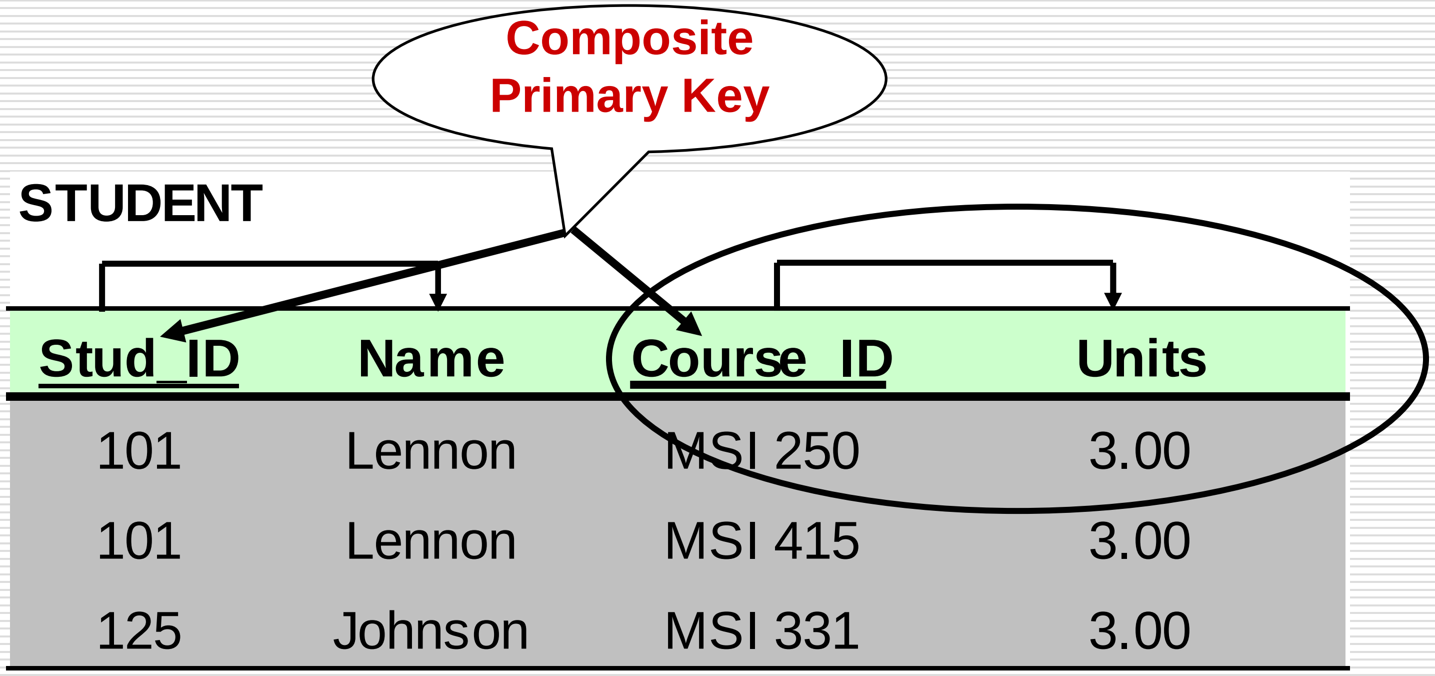


STUDENT

<u>**Stud_ID**</u>	**Name**	**Course_ID**	**Units**
101	Lennon	MSI 250	3.00
101	Lennon	MSI 415	3.00
125	Johnson	MSI 331	3.00

Bringing a Relation to 1NF

- Option 1: Make a determinant of the repeating group (or the multi-valued attribute) a part of the primary key.

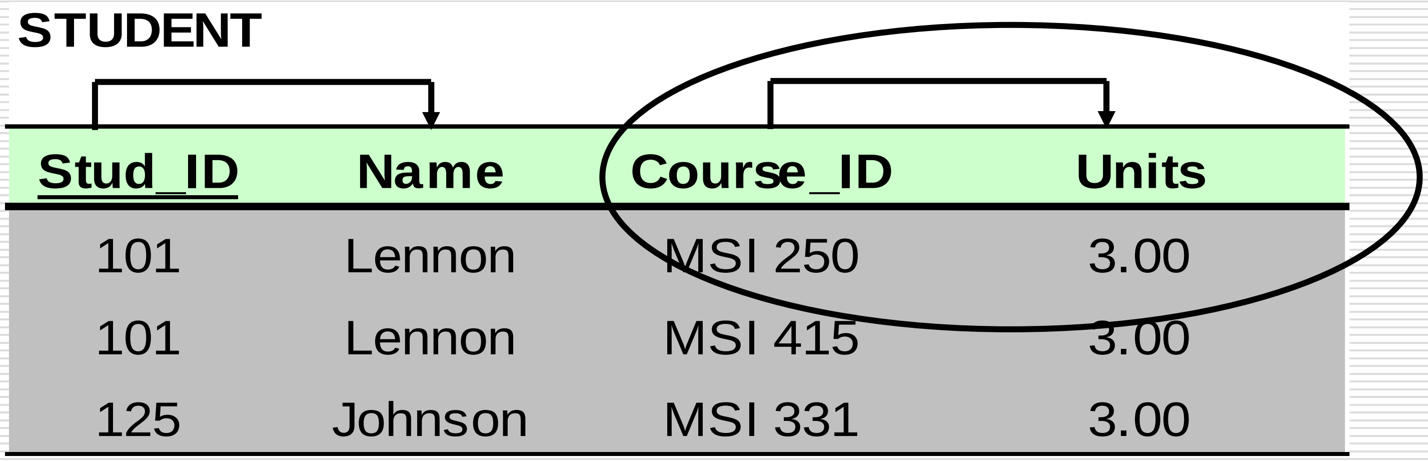


STUDENT

Stud_ID	Name	Course_ID	Units
101	Lennon	MSI 250	3.00
101	Lennon	MSI 415	3.00
125	Johnson	MSI 331	3.00

Bringing a Relation to 1NF

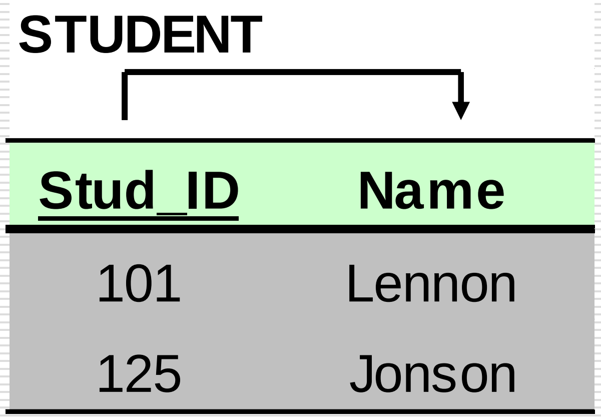
- Option 2: Remove the entire repeating group from the relation. Create another relation which would contain all the attributes of the repeating group, plus the primary key from the first relation. In this new relation, the primary key from the original relation and the determinant of the repeating group will comprise a primary key.



STUDENT

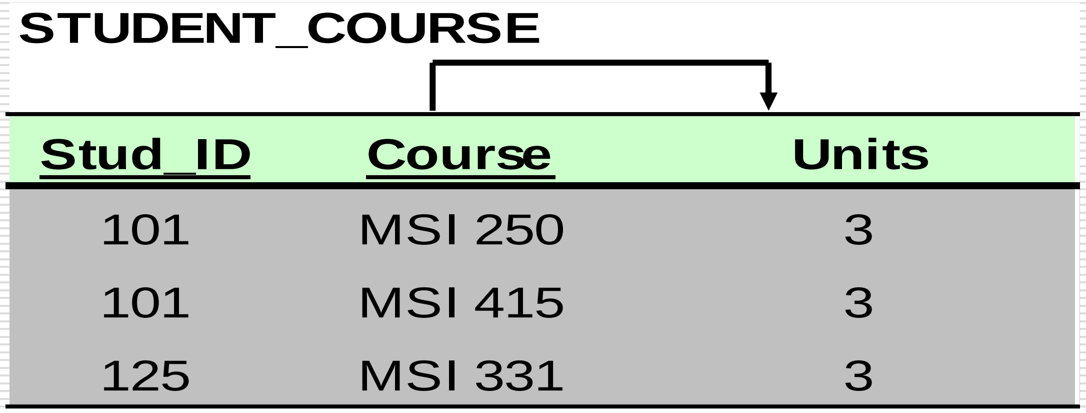
<u>Stud_ID</u>	**Name**	**Course_ID**	**Units**
101	Lennon	MSI 250	3.00
101	Lennon	MSI 415	3.00
125	Johnson	MSI 331	3.00

Bringing a Relation to 1NF



STUDENT

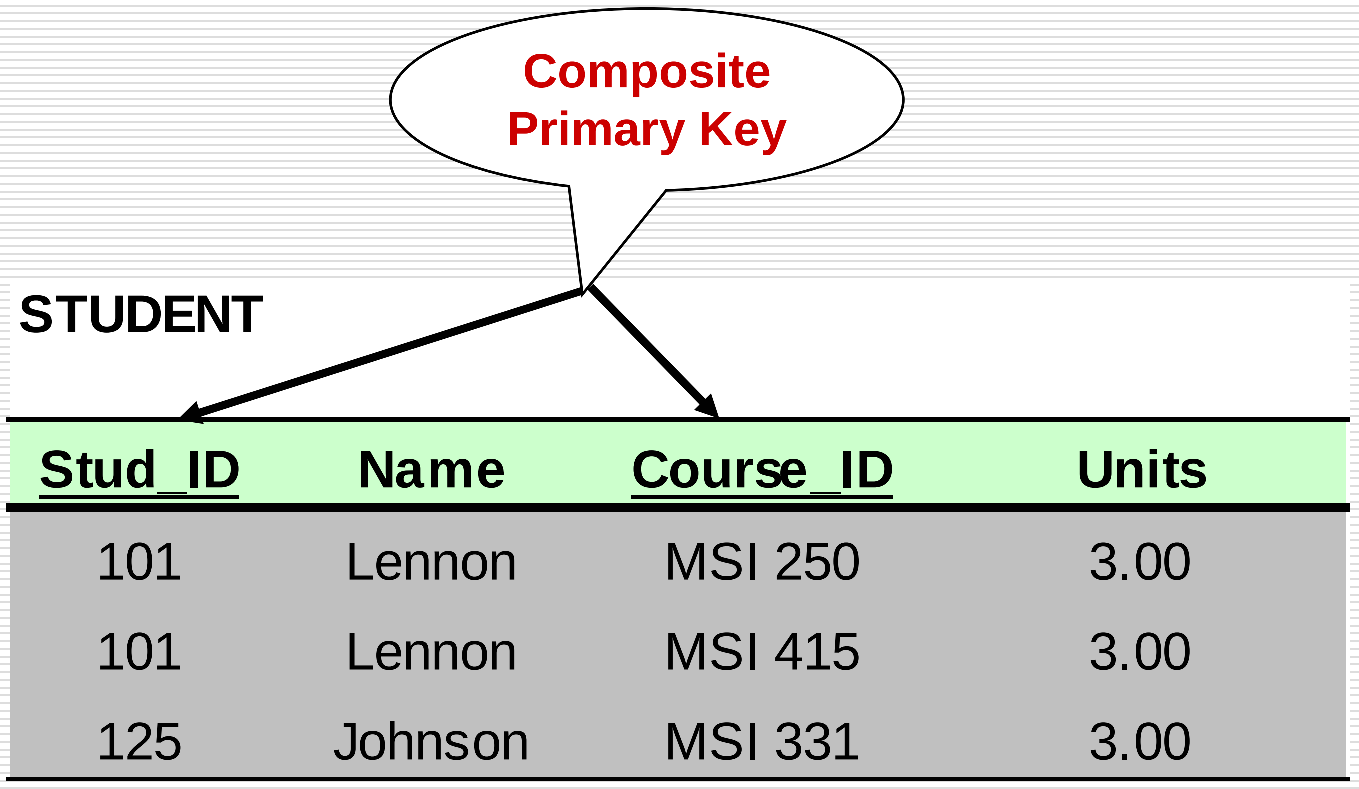
Stud_ID	**Name**
101	Lennon
125	Jonson



STUDENT_COURSE

Stud_ID	**Course**	**Units**
101	MSI 250	3
101	MSI 415	3
125	MSI 331	3

Bringing a Relation to 2NF

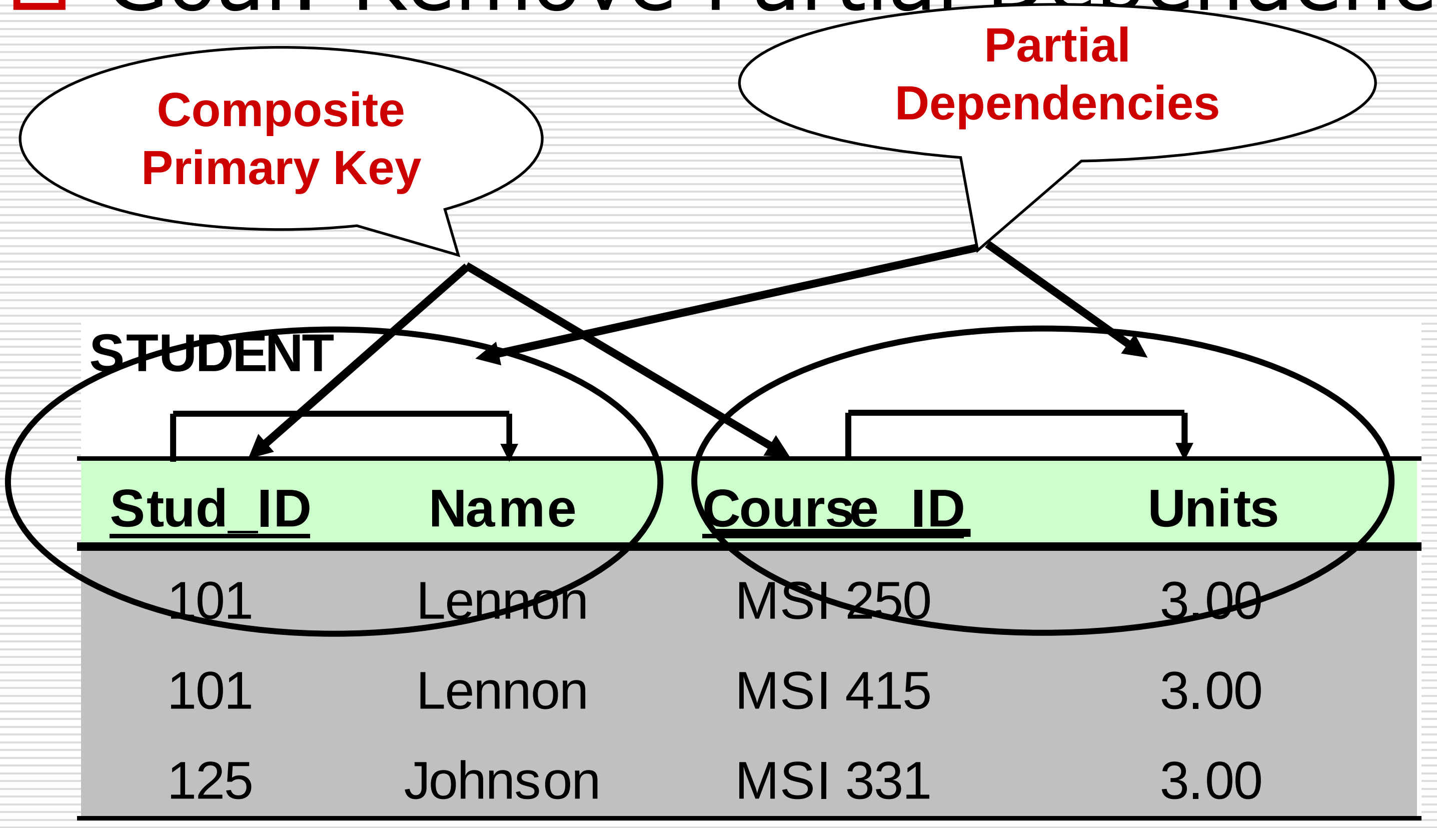


STUDENT

Stud_ID	Name	Course_ID	Units
101	Lennon	MSI 250	3.00
101	Lennon	MSI 415	3.00
125	Johnson	MSI 331	3.00

Bringing a Relation to 2NF

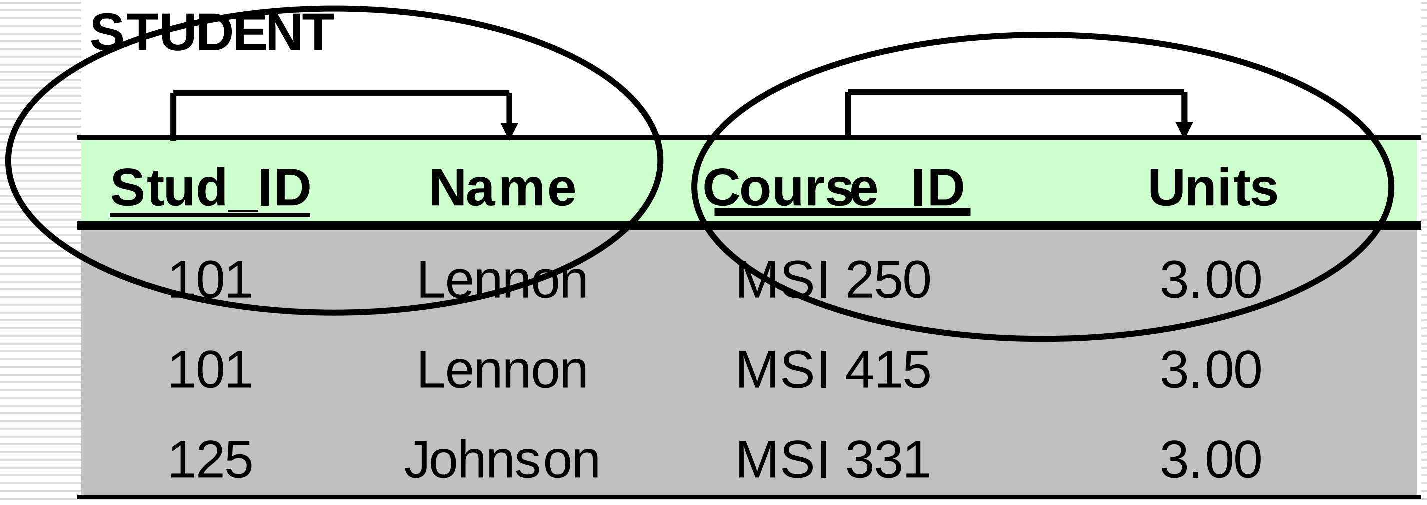
- Goal: Remove Partial Dependencies



Stud_ID	Name	Course_ID	Units
101	Lennon	MSI 250	3.00
101	Lennon	MSI 415	3.00
125	Johnson	MSI 331	3.00

Bringing a Relation to 2NF

- Remove attributes that are dependent from the part but not the whole of the primary key from the original relation. For each partial dependency, create a new relation, with the corresponding part of the primary key from the original as the primary key.



STUDENT

Stud_ID	**Name**	**Course_ID**	**Units**
101	Lennon	MSI 250	3.00
101	Lennon	MSI 415	3.00
125	Johnson	MSI 331	3.00

Bringing a Relation to 2NF

CUSTOMER

Stud_ID	Name	Course_ID	Units
101	Lennon	MSI 250	3.00
101	Lennon	MSI 415	3.00
125	Johnson	MSI 331	3.00

STUDENT_COURSE

Stud_ID	Course_ID
101	MSI 250
101	MSI 415
125	MSI 331

STUDENT

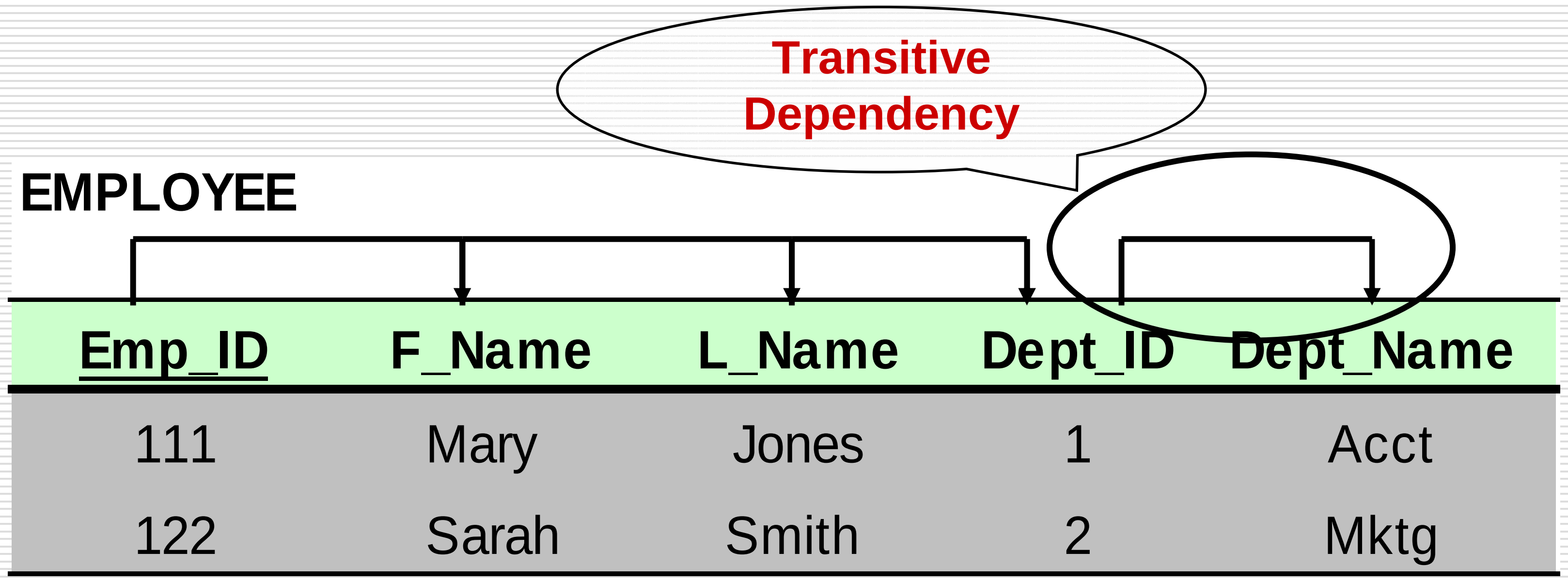
Stud_ID	Name
101	Lennon
101	Lennon
125	Johnson

COURSE

Course_ID	Units
MSI 250	3.00
MSI 415	3.00
MSI 331	3.00

Bringing a Relation to 3NF

- Goal: Get rid of transitive dependencies.

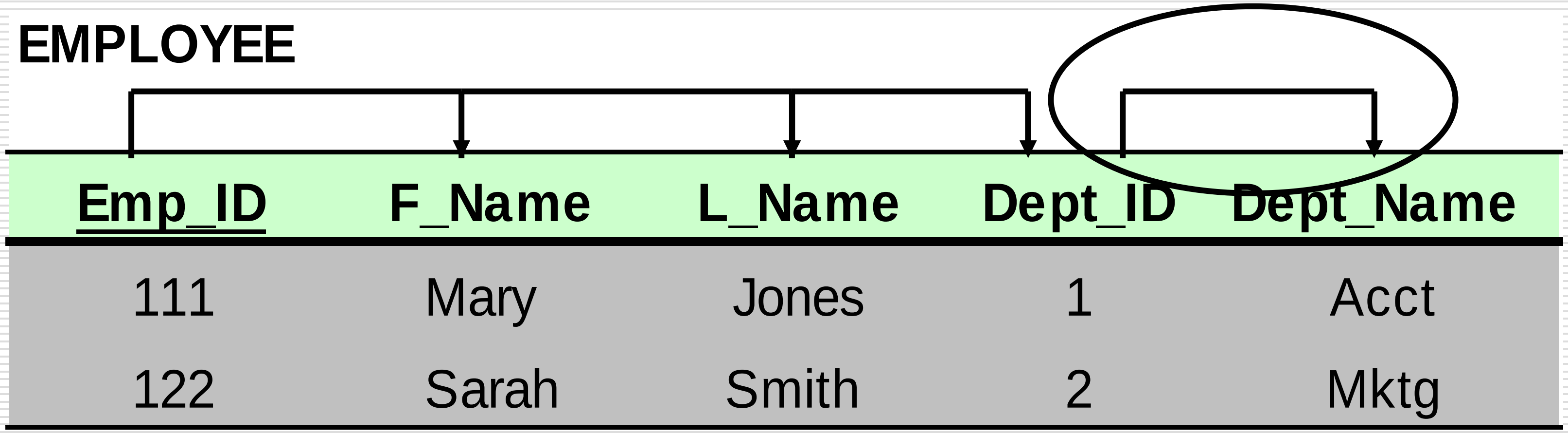


EMPLOYEE

Emp_ID	F_Name	L_Name	Dept_ID	Dept_Name
111	Mary	Jones	1	Acct
122	Sarah	Smith	2	Mktg

Bringing a Relation to 3NF

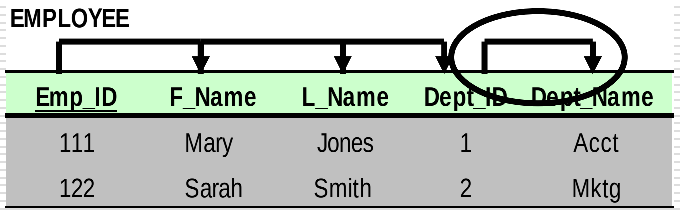
- Remove the attributes, which are dependent on a non-key attribute, from the original relation. For each transitive dependency, create a new relation with the non-key attribute which is a determinant in the transitive dependency as a primary key, and the dependent non-key attribute as a dependent.



EMPLOYEE

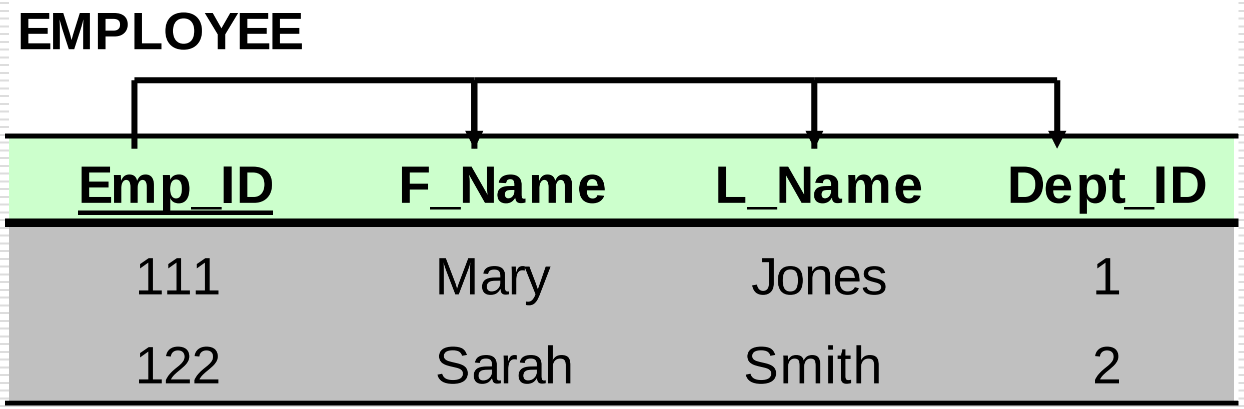
Emp_ID	F_Name	L_Name	Dept_ID	Dept_Name
111	Mary	Jones	1	Acct
122	Sarah	Smith	2	Mktg

Bringing a Relation to 3NF



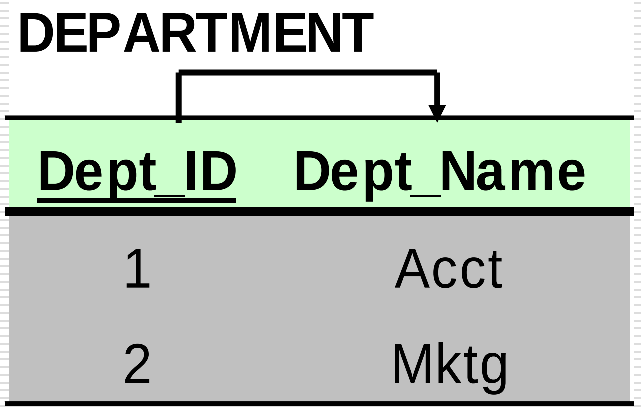
EMPLOYEE

<u>Emp_ID</u>	F_Name	L_Name	Dept_ID	Dept_Name
111	Mary	Jones	1	Acct
122	Sarah	Smith	2	Mktg



EMPLOYEE

<u>Emp_ID</u>	F_Name	L_Name	Dept_ID
111	Mary	Jones	1
122	Sarah	Smith	2



DEPARTMENT

<u>Dept_ID</u>	Dept_Name
1	Acct
2	Mktg