

# Chapter 5:

# Logical Database Design and the Relational Model

# Objectives

- Definition of terms
- List five properties of relations
- State two properties of candidate keys
- Define first, second, and third normal form
- Describe problems from merging relations
- Transform E-R and EER diagrams to relations
- Create tables with entity and relational integrity constraints
- Use normalization to convert anomalous tables to well-structured relations

# Relation

- Definition: A relation is a named, two-dimensional table of data
- Table consists of rows (records) and columns (attribute or field)
- Requirements for a table to qualify as a relation:
  - It must have a unique name
  - Every attribute value must be atomic (not multivalued, not composite)
  - Every row must be unique (can't have two rows with exactly the same values for all their fields)
  - Attributes (columns) in tables must have unique names
  - The order of the columns must be irrelevant
  - The order of the rows must be irrelevant

NOTE: all *relations* are in ***1<sup>st</sup> Normal form***

# Correspondence with E-R Model

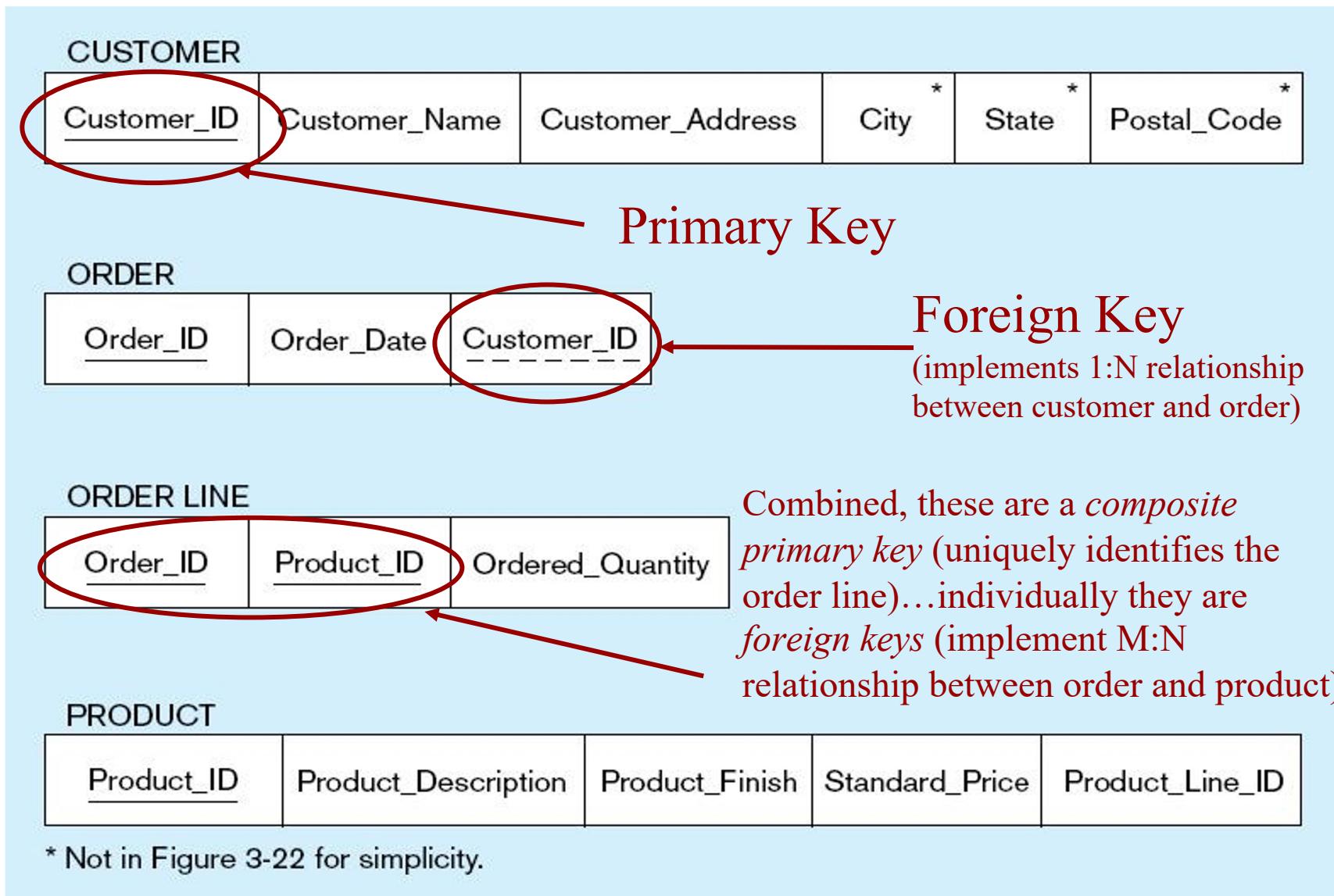
- Relations (tables) correspond with entity types and with many-to-many relationship types
- Rows correspond with entity instances and with many-to-many relationship instances
- Columns correspond with attributes
- NOTE: The word ***relation*** (in relational database) is NOT the same as the word ***relationship*** (in E-R model)

# Key Fields

- Keys are special fields that serve two main purposes:
  - **Primary keys** are unique identifiers of the relation in question. Examples include employee numbers, social security numbers, etc. *This is how we can guarantee that all rows are unique*
  - **Foreign keys** are identifiers that enable a dependent relation (on the many side of a relationship) to refer to its parent relation (on the one side of the relationship)
- Keys can be **simple** (a single field) or **composite** (more than one field)
- Keys usually are used as indexes to speed up the response to user queries (More on this in Chapter 6)



Figure 5-3 Schema for four relations (Pine Valley Furniture Company)



# Integrity Constraints

- Domain Constraints
  - Allowable values for an attribute. See Table 5-1
- Entity Integrity
  - No primary key attribute may be null. All primary key fields **MUST** have data
- Action Assertions
  - Business rules. Recall from Chapter 4

## 1. Domain Constraints

**Definition:** Domain constraints specify that each attribute in a database must have a definite range and type of values. These constraints restrict the kind of values that can be stored in a column. For example, a gender column may be constrained to accept only the values "Male", "Female", or "Other".

**Purpose:** Ensures that data entries conform to the predefined data types and range limitations, preventing invalid data from being stored.

## 2. Entity Integrity Constraints

**Definition:** This constraint ensures that each row (or record) in a table has a unique identity, which is typically enforced by the primary key. No part of a primary key may be null because a primary key value is used to identify individual rows.

**Purpose:** Maintains the uniqueness of each row in a table, ensuring that each record can be uniquely identified, which is critical for the accurate retrieval and manipulation of data.

## Referential Integrity Constraints

**Definition:** Referential integrity is concerned with maintaining consistency among rows between two related tables. This constraint ensures that a foreign key value in one table either matches a primary key value in another table or is null. This maintains the validity of the links among the tables.

**Purpose:** Prevents "orphan" rows in the database by ensuring that relationships between tables remain consistent. For example, if a row in a Customer table is deleted, referential integrity could prevent this action if there are related orders in an Orders table unless those orders are also deleted or assigned to another customer.

## 4. Check Constraints

**Definition:** Check constraints allow specifying a condition on a column that must be true for each row in the table. For example, a constraint might require that the value in an age column must be greater than 18.

**Purpose:** Provides a way to enforce additional rules on data in a table beyond domain constraints, ensuring data meets business rules specifications.

## 5. Unique Constraints

**Definition:** Ensures that all values in a column are different from one another. This constraint is similar to a primary key but does not necessarily have to be the identifier for a row.

**Purpose:** Prevents duplicate entries in certain fields that require uniqueness, such as a user's email address or a registration number.

**Table 5-1** Domain Definitions for INVOICE Attributes

ATTRIBUTE	DOMAIN NAME	DESCRIPTION	DOMAIN
Customer_ID	Customer_IDs	Set of all possible customer IDs	character: size 5
Customer_Name	Customer_Names	Set of all possible customer names	character: size 25
Customer_Address	Customer_Addresses	Set of all possible customer addresses	character: size 30
City	Cities	Set of all possible cities	character: size 20
State	States	Set of all possible states	character: size 2
Postal_Code	Postal_Codes	Set of all possible postal zip codes	character: size 10
Order_ID	Order_IDs	Set of all possible order IDs	character: size 5
Order_Date	Order_Dates	Set of all possible order dates	date format mm/dd/yy
Product_ID	Product_IDs	Set of all possible product IDs	character: size 5
Product_Description	Product_Descriptions	Set of all possible product descriptions	character size 25
Product_Finish	Product_Finishes	Set of all possible product finishes	character: size 15
Standard_Price	Unit_Prices	Set of all possible unit prices	monetary: 6 digits
Product_Line_ID	Product_Line_IDs	Set of all possible product line IDs	integer: 3 digits
Ordered_Quantity	Quantities	Set of all possible ordered quantities	integer: 3 digits

Domain definitions enforce domain integrity constraints

# Integrity Constraints

- Referential Integrity—rule states that any foreign key value (on the relation of the many side) MUST match a primary key value in the relation of the one side. (Or the foreign key can be null)
  - For example: Delete Rules
    - Restrict—don't allow delete of “parent” side if related rows exist in “dependent” side
    - Cascade—automatically delete “dependent” side rows that correspond with the “parent” side row to be deleted
    - Set-to-Null—set the foreign key in the dependent side to null if deleting from the parent side □ not allowed for weak entities

## Figure 5-5 Referential integrity constraints (Pine Valley Furniture)

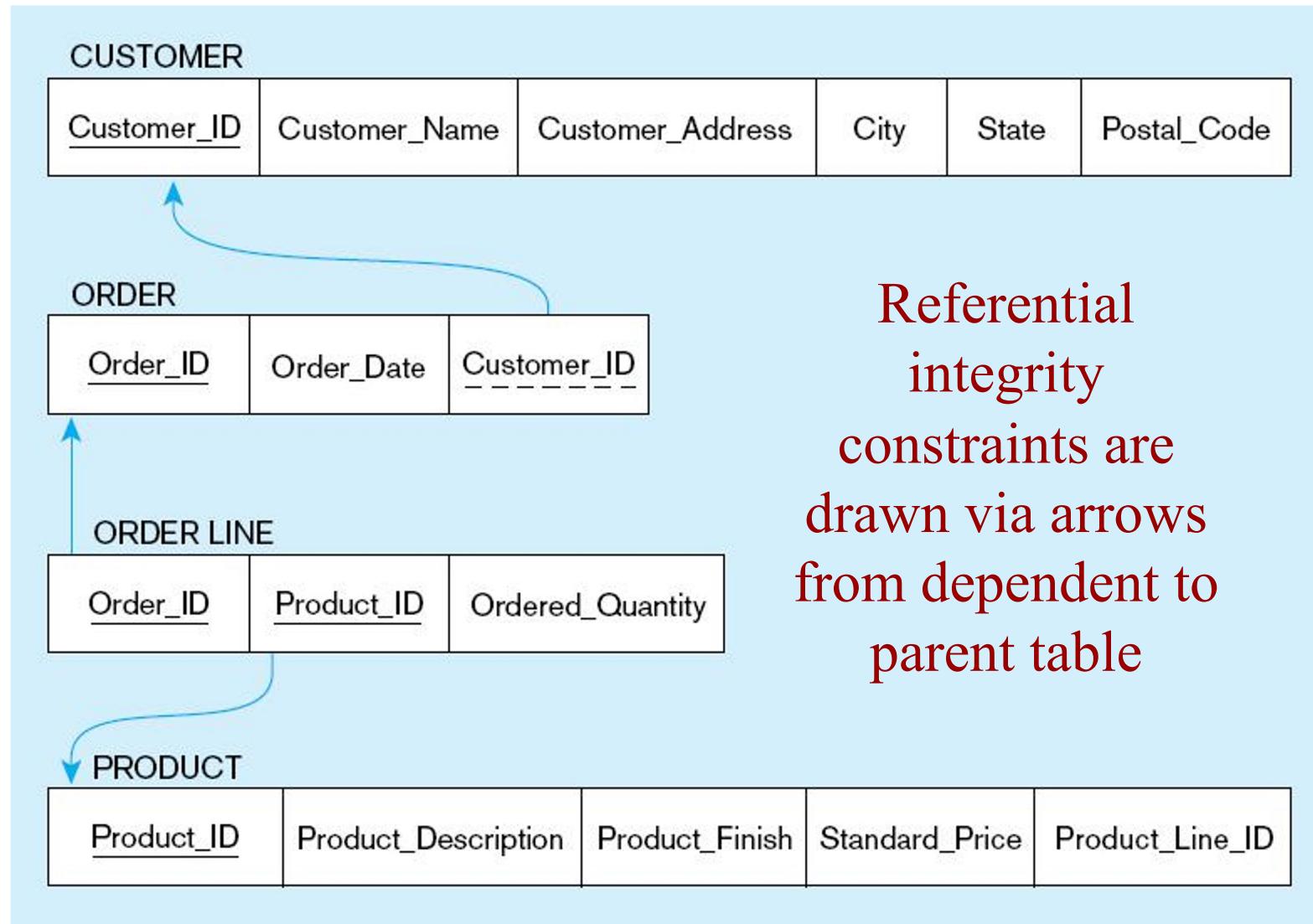


Figure 5-6 SQL table definitions

```
CREATE TABLE CUSTOMER
  (CUSTOMER_ID           VARCHAR(5)      NOT NULL,
   CUSTOMER_NAME         VARCHAR(25)     NOT NULL,
   CUSTOMER_ADDRESS      VARCHAR(30)     NOT NULL,
   CITY                  VARCHAR(20)      NOT NULL,
   STATE                 CHAR(2)        NOT NULL,
   POSTAL_CODE           CHAR(10)       NOT NULL,
   PRIMARY KEY (CUSTOMER_ID));

CREATE TABLE ORDER
  (ORDER_ID              CHAR(5)        NOT NULL,
   ORDER_DATE             DATE          NOT NULL,
   CUSTOMER_ID            VARCHAR(5)     NOT NULL,
   PRIMARY KEY (ORDER_ID),
   FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER (CUSTOMER_ID));

CREATE TABLE ORDER_LINE
  (ORDER_ID               CHAR(5)        NOT NULL,
   PRODUCT_ID             CHAR(5)        NOT NULL,
   ORDERED_QUANTITY       INT           NOT NULL,
   PRIMARY KEY (ORDER_ID, PRODUCT_ID),
   FOREIGN KEY (ORDER_ID) REFERENCES ORDER (ORDER_ID),
   FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT (PRODUCT_ID));

CREATE TABLE PRODUCT
  (PRODUCT_ID             CHAR(5)        NOT NULL,
   PRODUCT_DESCRIPTION    VARCHAR(25),
   PRODUCT_FINISH         VARCHAR(12),
   STANDARD_PRICE         DECIMAL(8,2),
   PRODUCT_LINE_ID        INT           NOT NULL,
   PRIMARY KEY (PRODUCT_ID));
```

Referential integrity constraints are implemented with foreign key to primary key references

# Transforming EER Diagrams into Relations

## Mapping Regular Entities to Relations

1. Simple attributes: E-R attributes map directly onto the relation
2. Composite attributes: Use only their simple, component attributes
3. Multivalued Attribute: Becomes a separate relation with a foreign key taken from the superior entity

Figure 5-8 Mapping a regular entity

**(a) CUSTOMER entity type with simple attributes**

CUSTOMER
<u>Customer_ID</u>
Customer_Name
Customer_Address
Postal_Code

**(b) CUSTOMER relation**

CUSTOMER			
<u>Customer_ID</u>	Customer_Name	Customer_Address	Postal_Code

## Figure 5-9 Mapping a composite attribute

(a) CUSTOMER entity type with composite attribute

CUSTOMER
<u>Customer_ID</u>
Customer_Name
Customer_Address
(Street, City, State)
Postal_Code

(b) CUSTOMER relation with address detail

CUSTOMER					
<u>Customer_ID</u>	Customer_Name	Street	City	State	Postal_Code

Figure 5-10 Mapping an entity with a multivalued attribute

(a)

EMPLOYEE
<u>Employee_ID</u>
Employee_Name
Employee_Address
{Skill}

Multivalued attribute becomes a separate relation with foreign key

(b)

EMPLOYEE		
<u>Employee_ID</u>	Employee_Name	Employee_Address

EMPLOYEE_SKILL	
<u>Employee_ID</u>	Skill

One-to-many relationship between original entity and new relation

# Transforming EER Diagrams into Relations (cont.)

## Mapping Weak Entities

- Becomes a separate relation with a foreign key taken from the superior entity
- Primary key composed of:
  - Partial identifier of weak entity
  - Primary key of identifying relation (strong entity)

### Definition of Weak Entities

Weak entities are types of entities in a database that do not have a primary key of their own and cannot be uniquely identified by their attributes alone. They are dependent on another entity, known as a strong (or owner) entity, to ensure their uniqueness and existence within the database. This dependency is established through a relationship known as an identifying relationship.

### Characteristics of Weak Entities:

No sufficient attributes: Weak entities do not have attributes that can uniquely identify their instances.  
Dependency: They depend on some strong entity for their identity.

Discriminator (Partial Key): A weak entity typically has a discriminator or partial key, a set of attributes that, when combined with the primary key of the strong entity, uniquely identifies records in the weak entity.

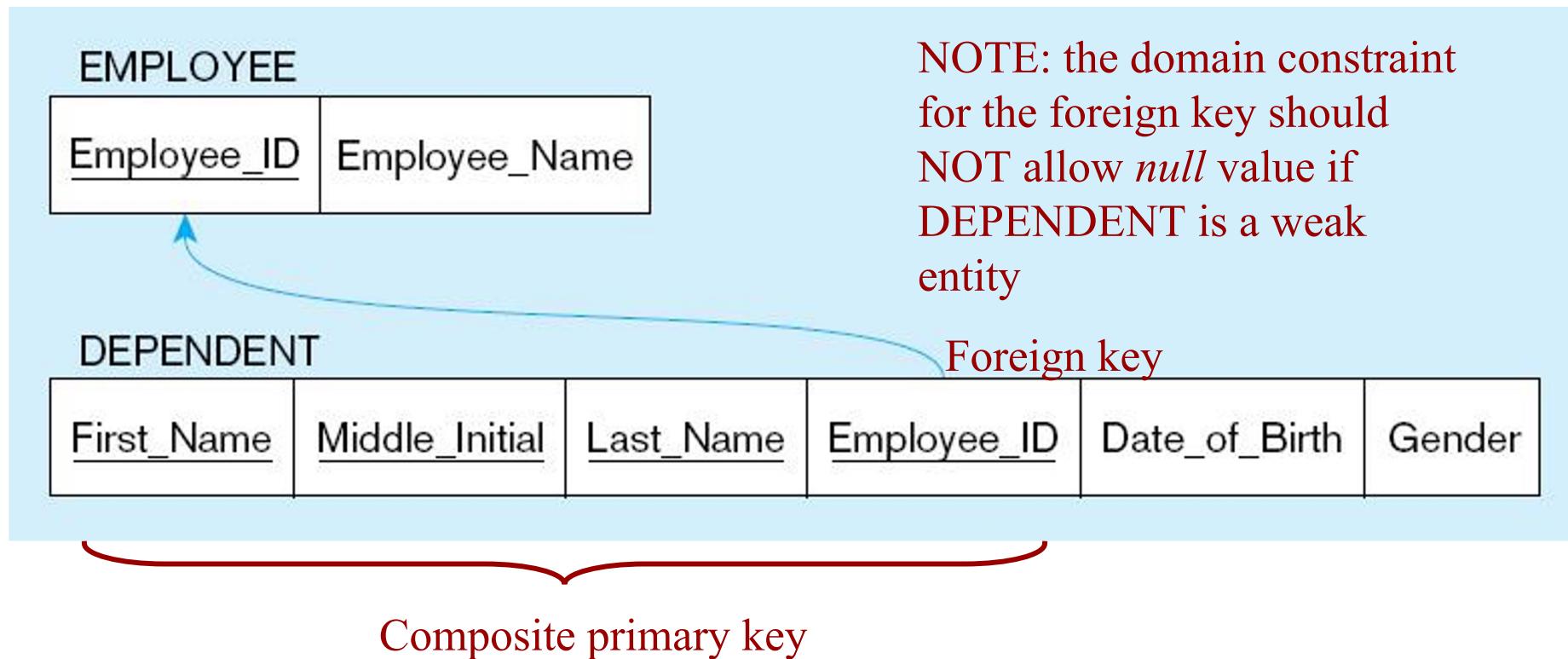
Figure 5-11 Example of mapping a weak entity

a) Weak entity DEPENDENT



Figure 5-11 Example of mapping a weak entity (cont.)

b) Relations resulting from weak entity



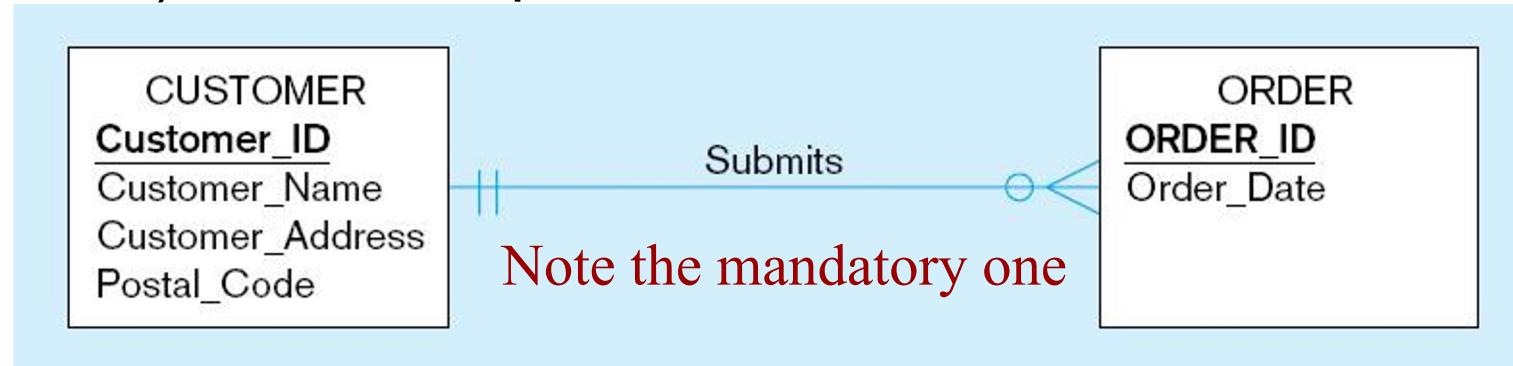
# Transforming EER Diagrams into Relations (cont.)

## Mapping Binary Relationships

- One-to-Many—Primary key on the one side becomes a foreign key on the many side
- Many-to-Many—Create a ***new relation*** with the primary keys of the two entities as its primary key
- One-to-One—Primary key on the mandatory side becomes a foreign key on the optional side

Figure 5-12 Example of mapping a 1:M relationship

a) Relationship between customers and orders



b) Mapping the relationship

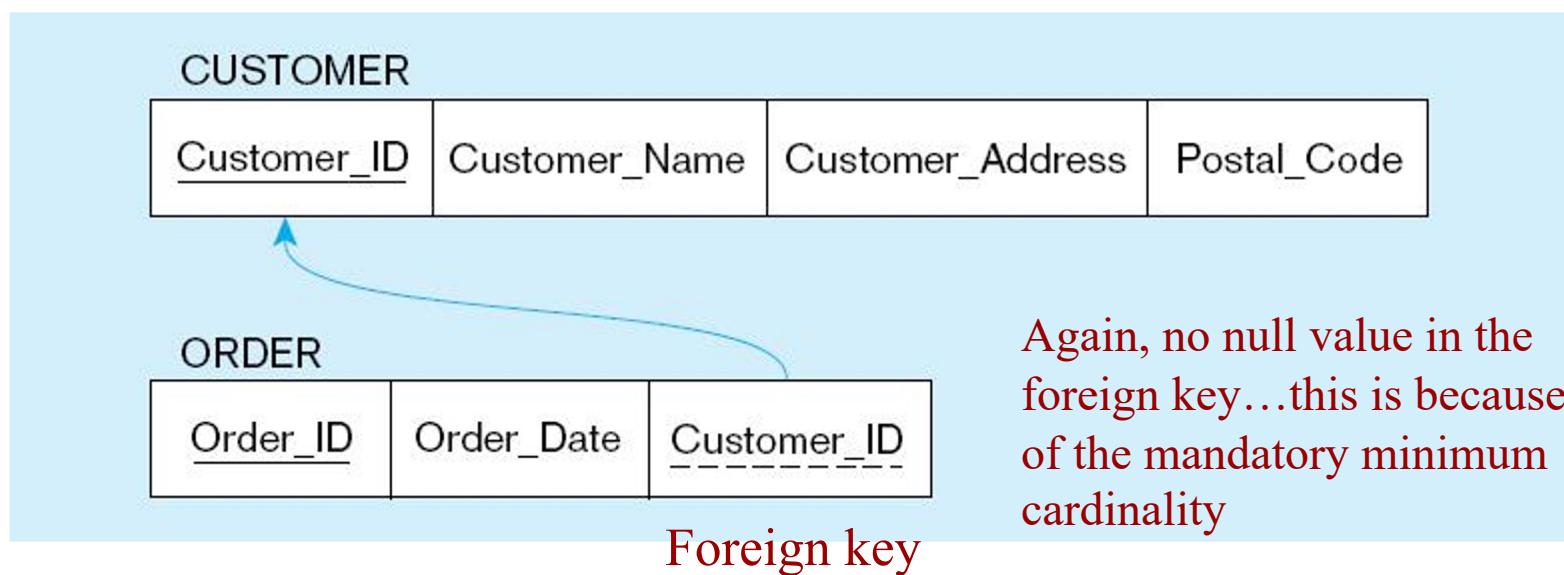
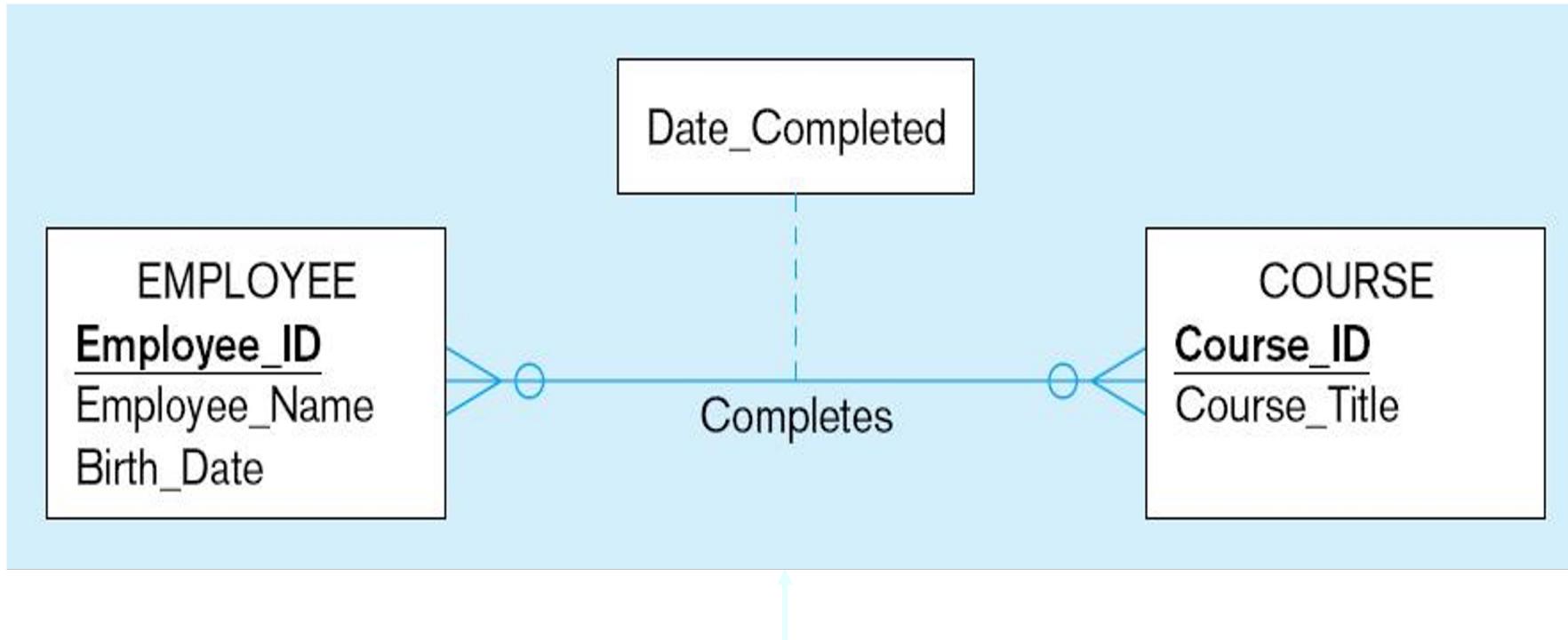


Figure 5-13 Example of mapping an M:N relationship

a) Completes relationship (M:N)



The *Completes* relationship will need to become a separate relation

Figure 5-13 Example of mapping an M:N relationship (cont.)

b) Three resulting relations

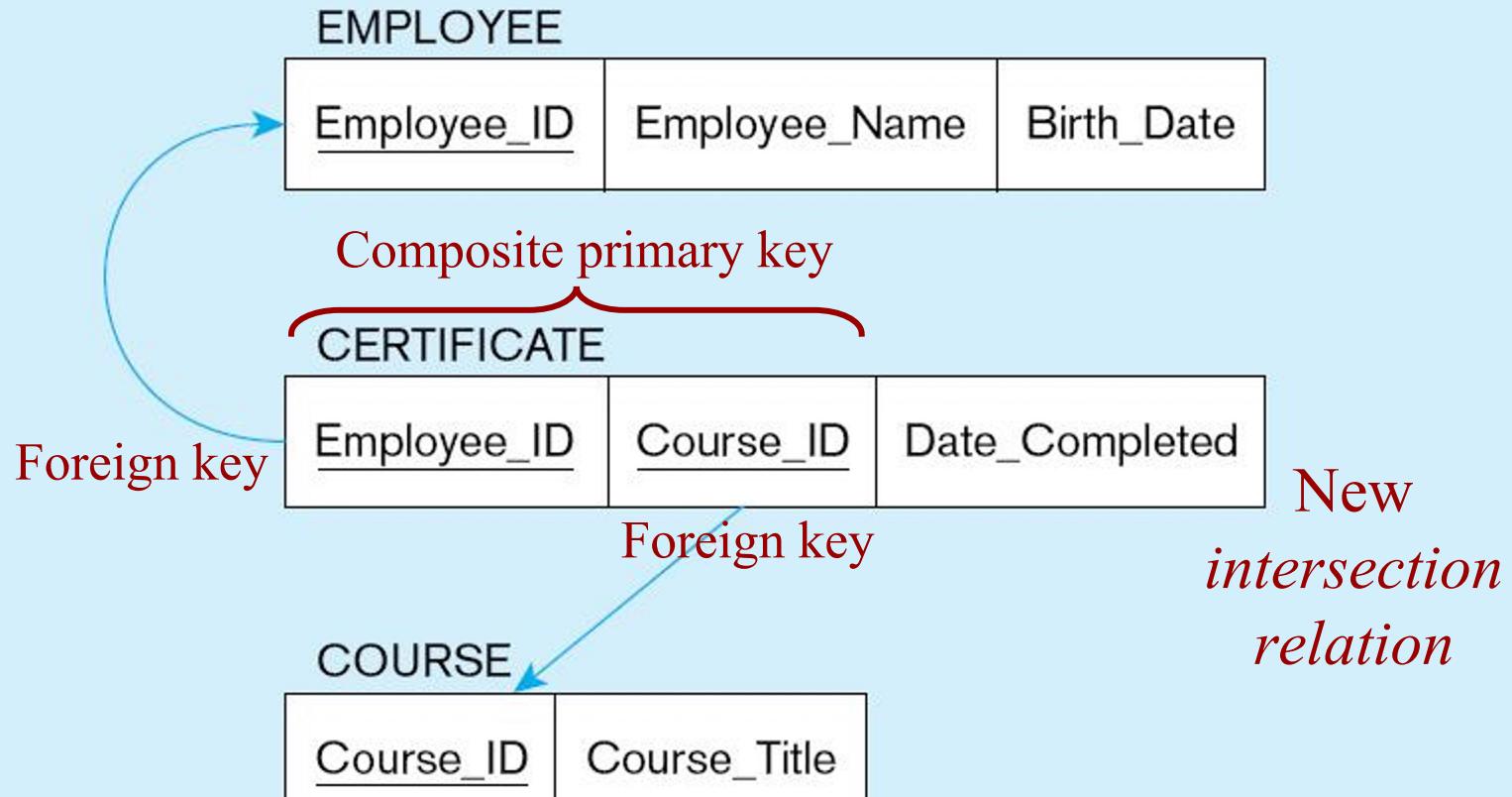
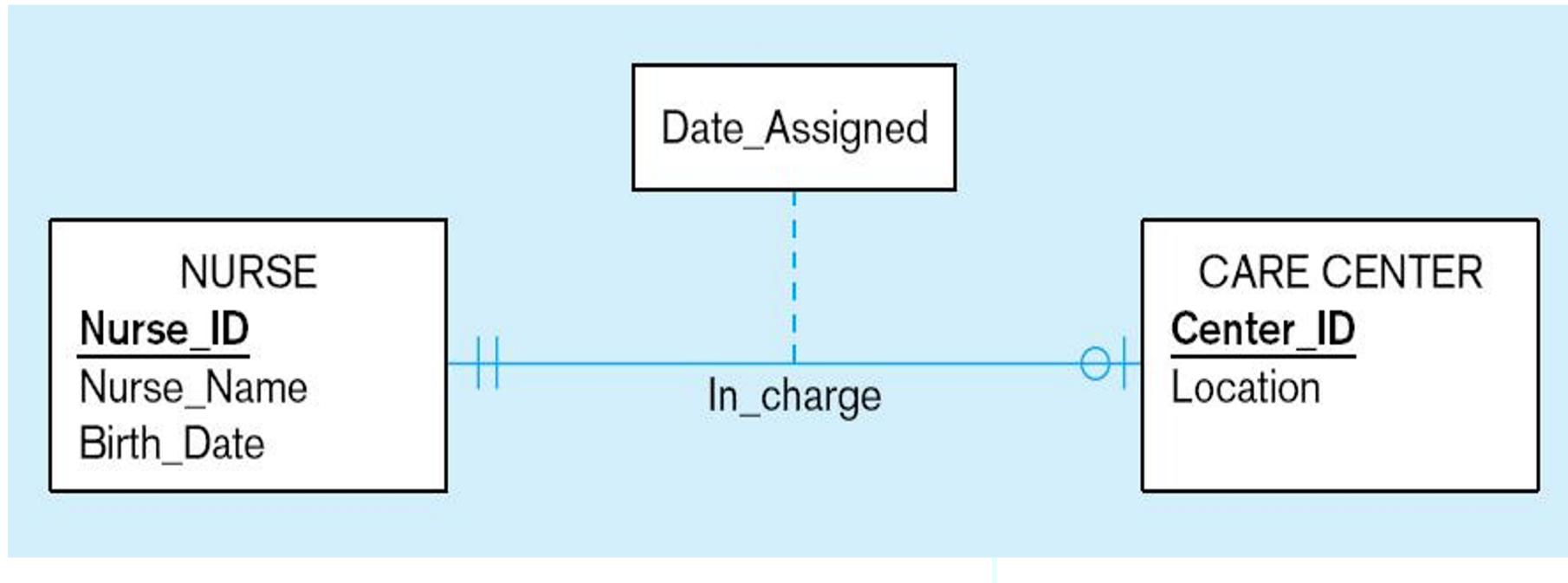


Figure 5-14 Example of mapping a binary 1:1 relationship

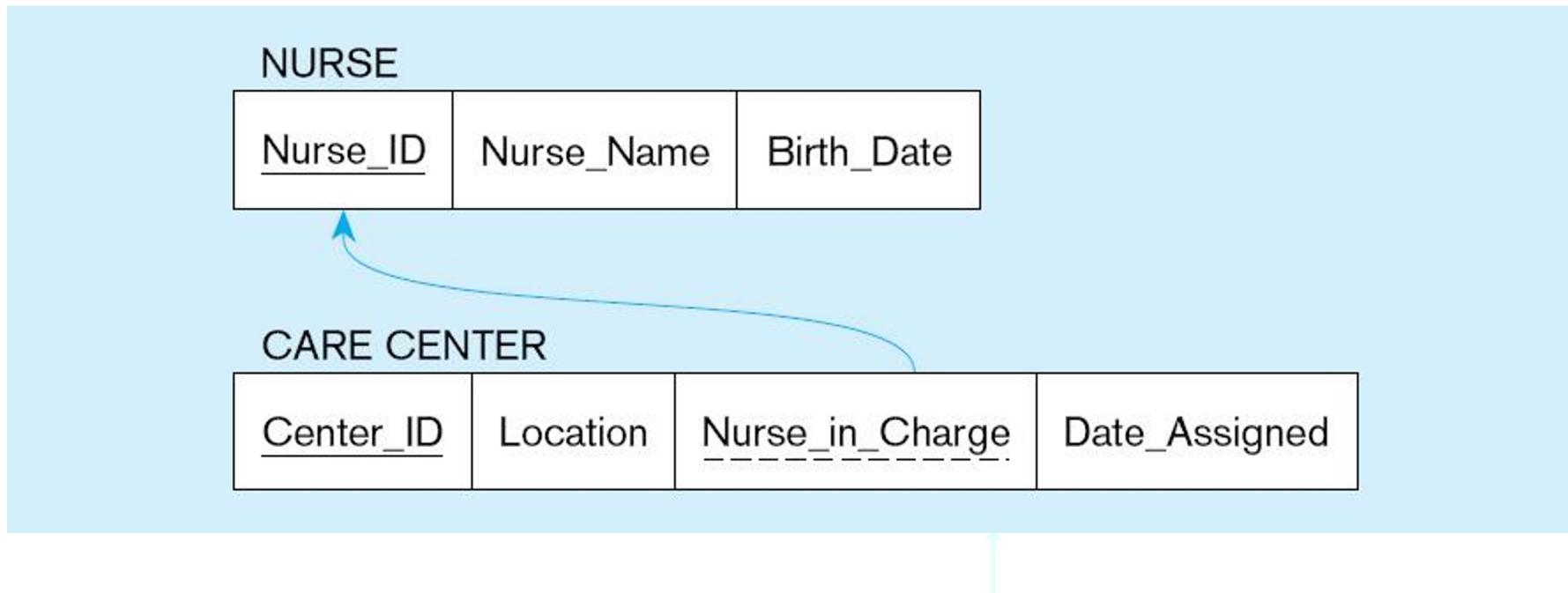
a) In\_charge relationship (1:1)



Often in 1:1 relationships, one direction is optional

Figure 5-14 Example of mapping a binary 1:1 relationship (cont.)

b) Resulting relations



Foreign key goes in the relation on the optional side,  
matching the primary key on the mandatory side

# Transforming EER Diagrams into Relations (cont.)

## Mapping Associative Entities

- Identifier Not Assigned
  - Default primary key for the association relation is composed of the primary keys of the two entities (as in M:N relationship)
- Identifier Assigned
  - It is natural and familiar to end-users
  - Default identifier may not be unique

## Figure 5-15 Example of mapping an associative entity

a) An associative  
entity

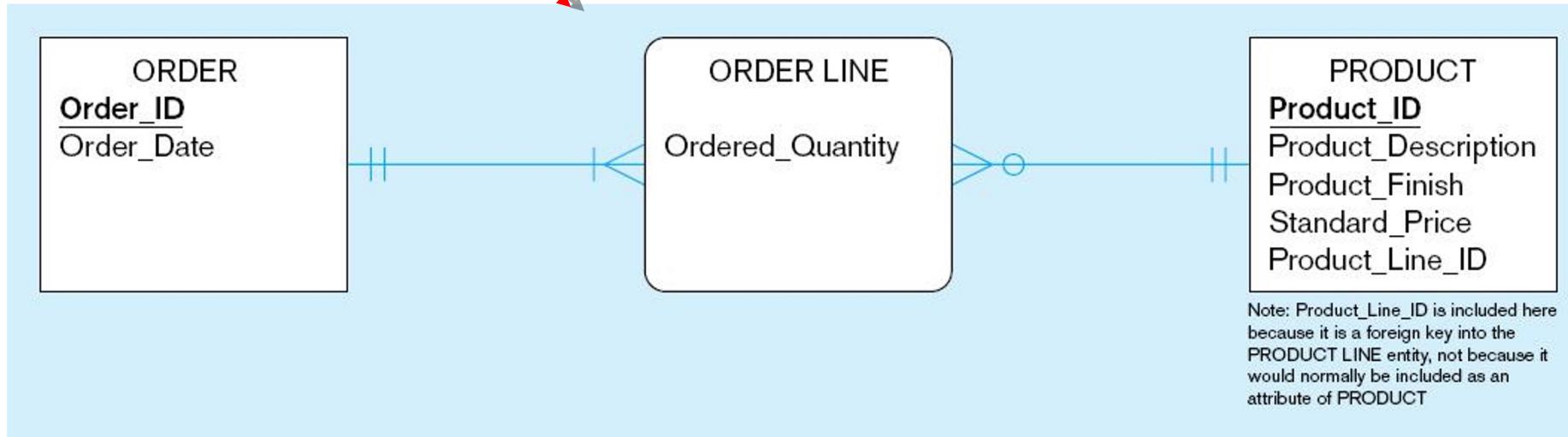
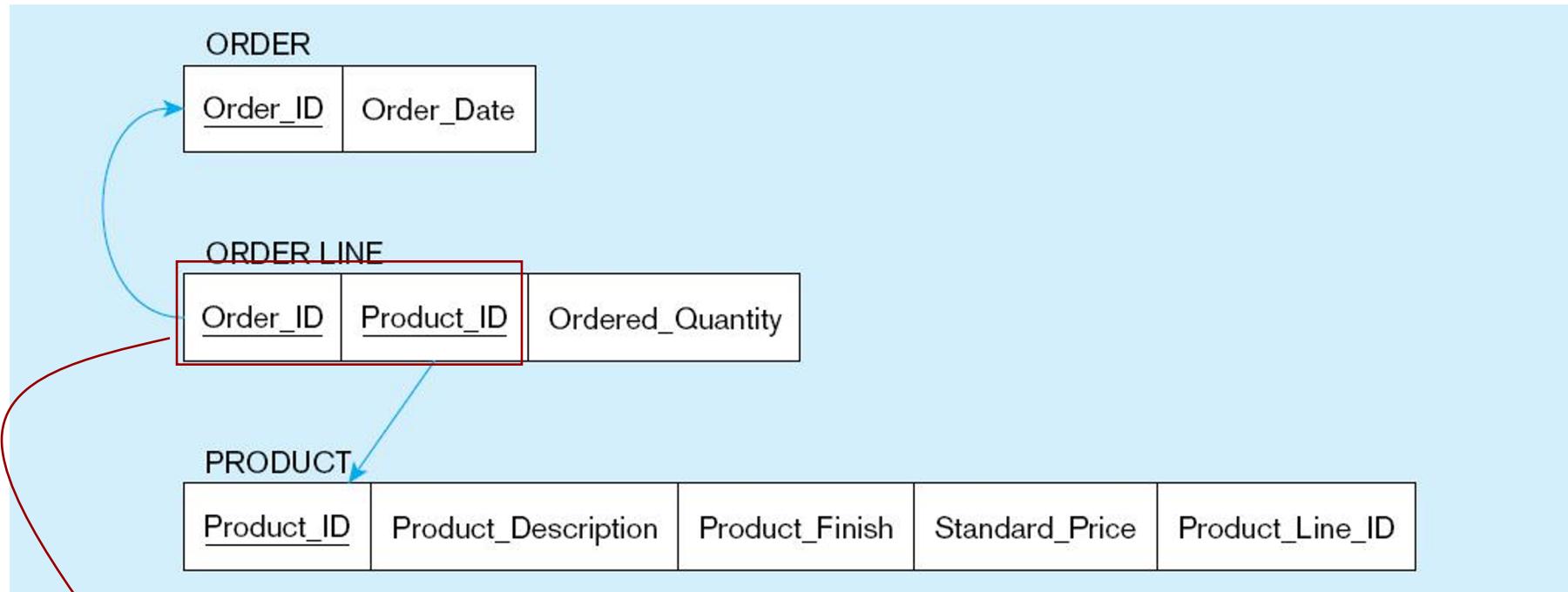


Figure 5-15 Example of mapping an associative entity (cont.)

b) Three resulting relations



Composite primary key formed from the two foreign keys

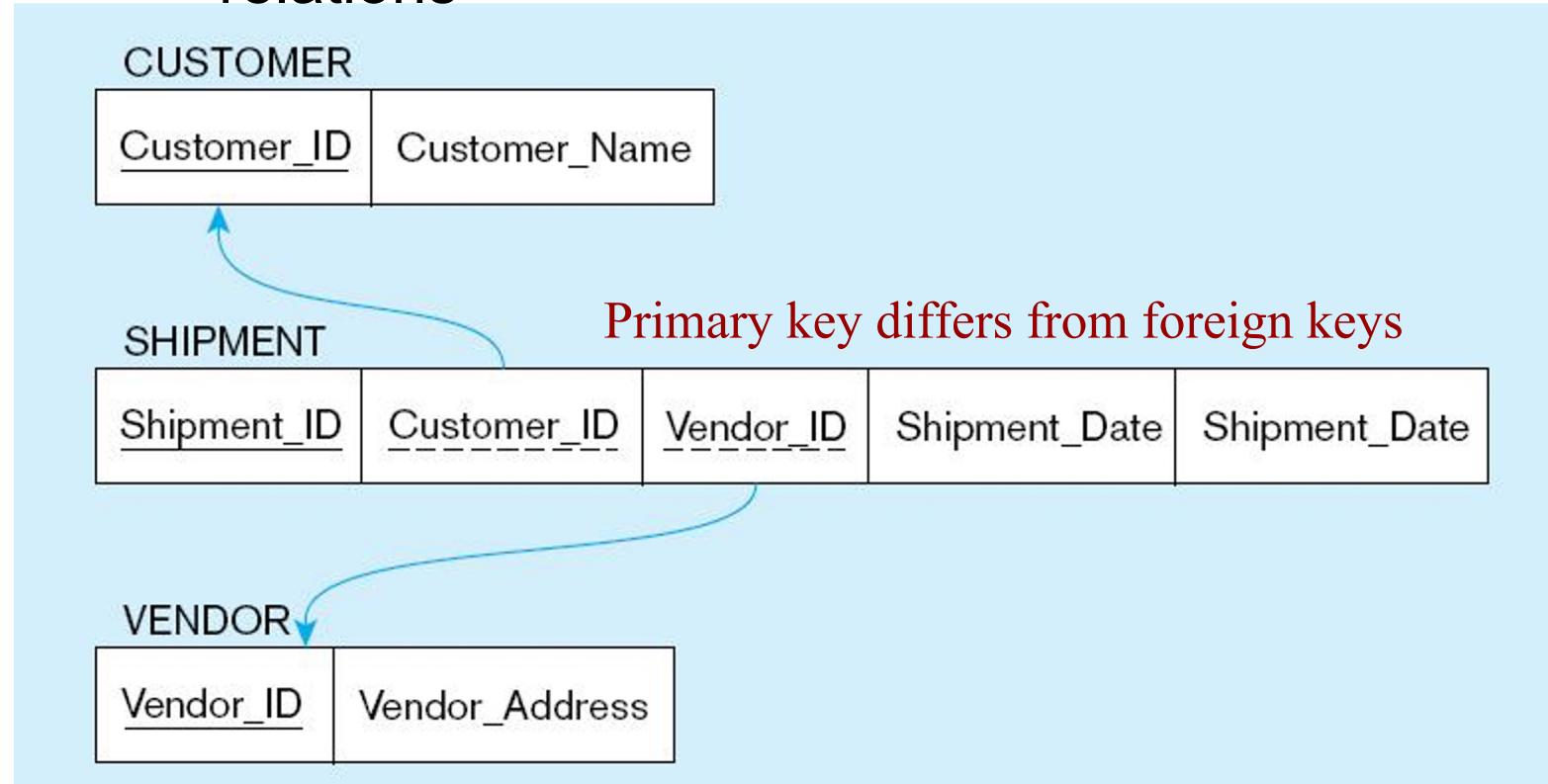
Figure 5-16 Example of mapping an associative entity with an identifier

a) SHIPMENT associative entity



Figure 5-16 Example of mapping an associative entity with an identifier (cont.)

b) Three resulting relations

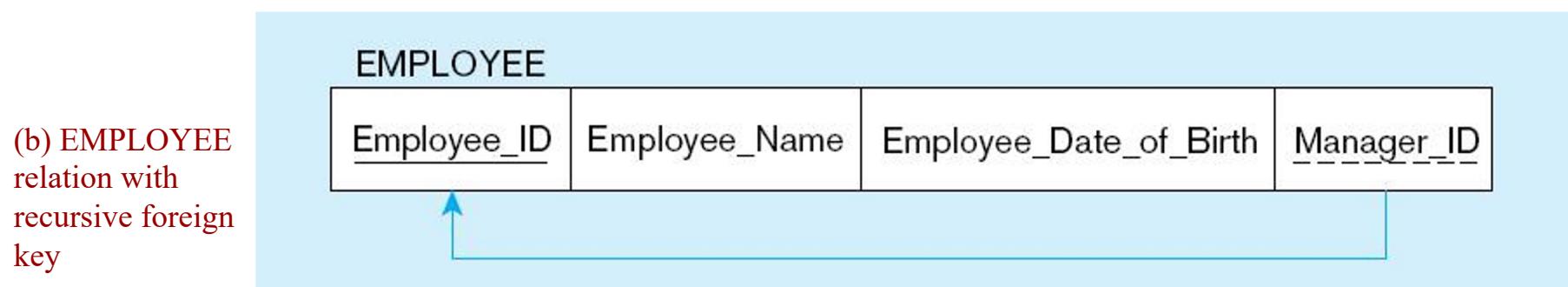
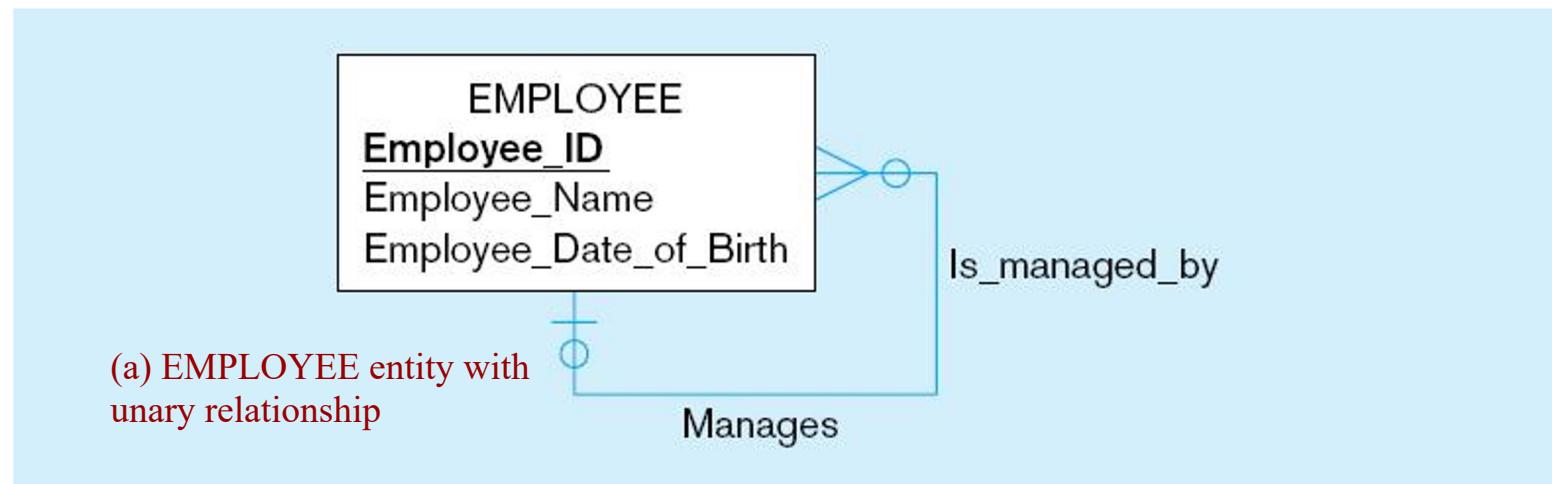


# Transforming EER Diagrams into Relations (cont.)

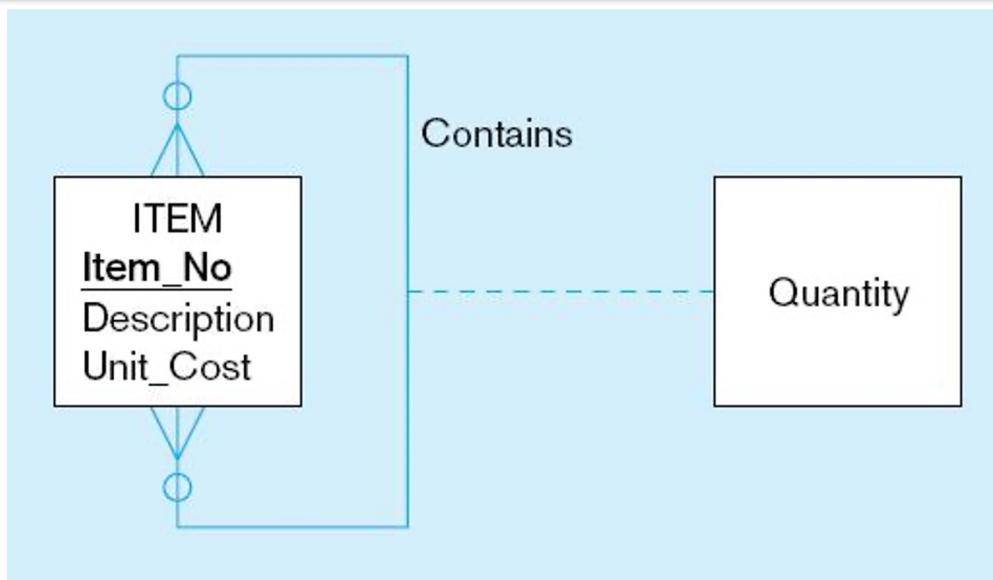
## Mapping Unary Relationships

- One-to-Many—Recursive foreign key in the same relation
- Many-to-Many—Two relations:
  - One for the entity type
  - One for an associative relation in which the primary key has two attributes, both taken from the primary key of the entity

## Figure 5-17 Mapping a unary 1:N relationship

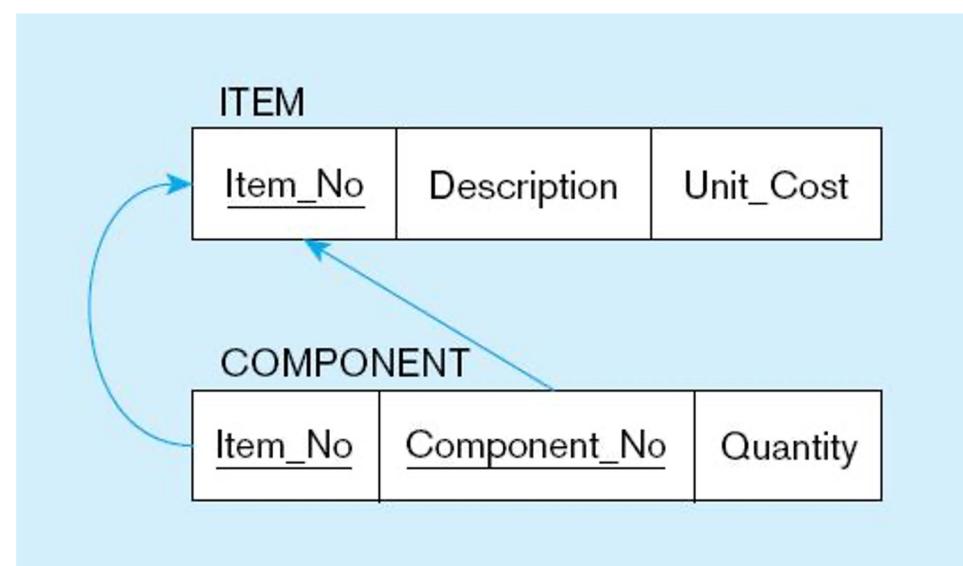


## Figure 5-18 Mapping a unary M:N relationship



(a) Bill-of-materials  
relationships (M:N)

(b) ITEM and  
COMPONENT  
relations



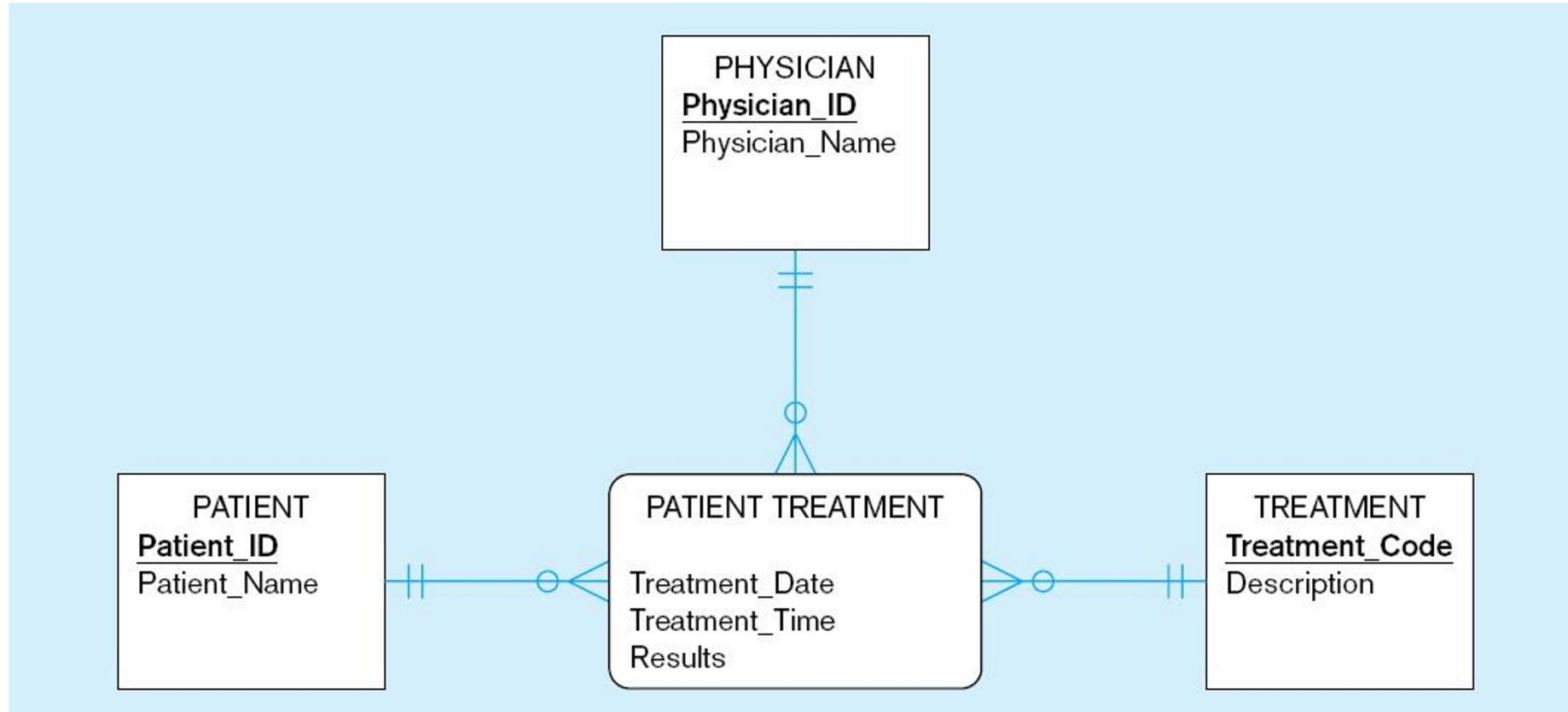
# Transforming EER Diagrams into Relations (cont.)

## Mapping Ternary (and n-ary) Relationships

- One relation for each entity and one for the associative entity
- Associative entity has foreign keys to each entity in the relationship

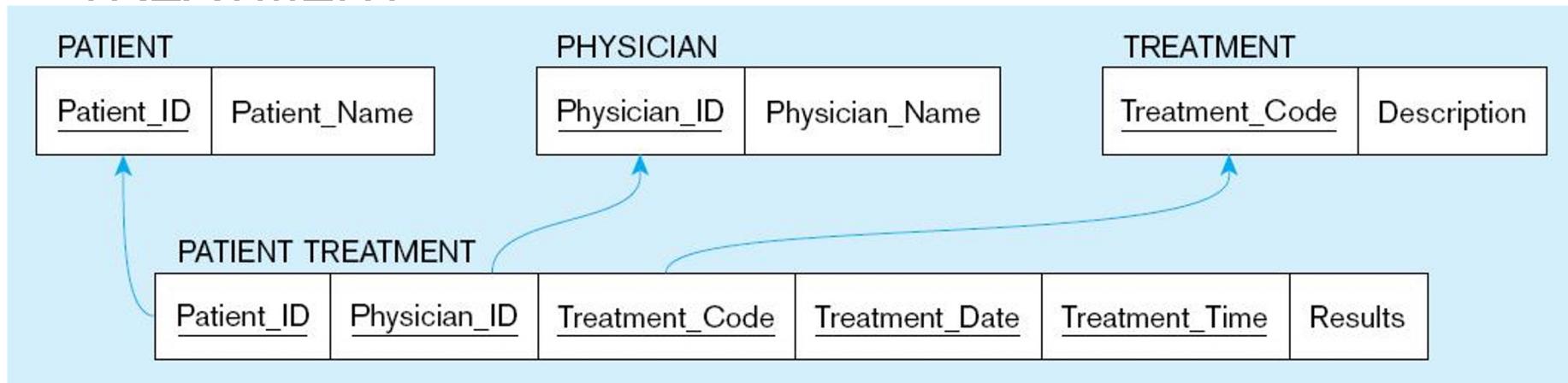
## Figure 5-19 Mapping a ternary relationship

a) PATIENT TREATMENT Ternary relationship with associative entity



## Figure 5-19 Mapping a ternary relationship (cont.)

### b) Mapping the ternary relationship PATIENT TREATMENT



Remember  
that the  
primary key  
MUST be  
unique

This is why  
treatment date  
and time are  
included in the  
composite  
primary key

But this makes a  
very  
cumbersome  
key...

It would be  
better to create a  
surrogate key  
like Treatment#

# Transforming EER Diagrams into Relations (cont.)

## Mapping Supertype/Subtype Relationships

- One relation for supertype and for each subtype
- Supertype attributes (including identifier and subtype discriminator) go into supertype relation
- Subtype attributes go into each subtype; primary key of supertype relation also becomes primary key of subtype relation
- 1:1 relationship established between supertype and each subtype, with supertype as primary table

## Figure 5-20 Supertype/subtype relationships

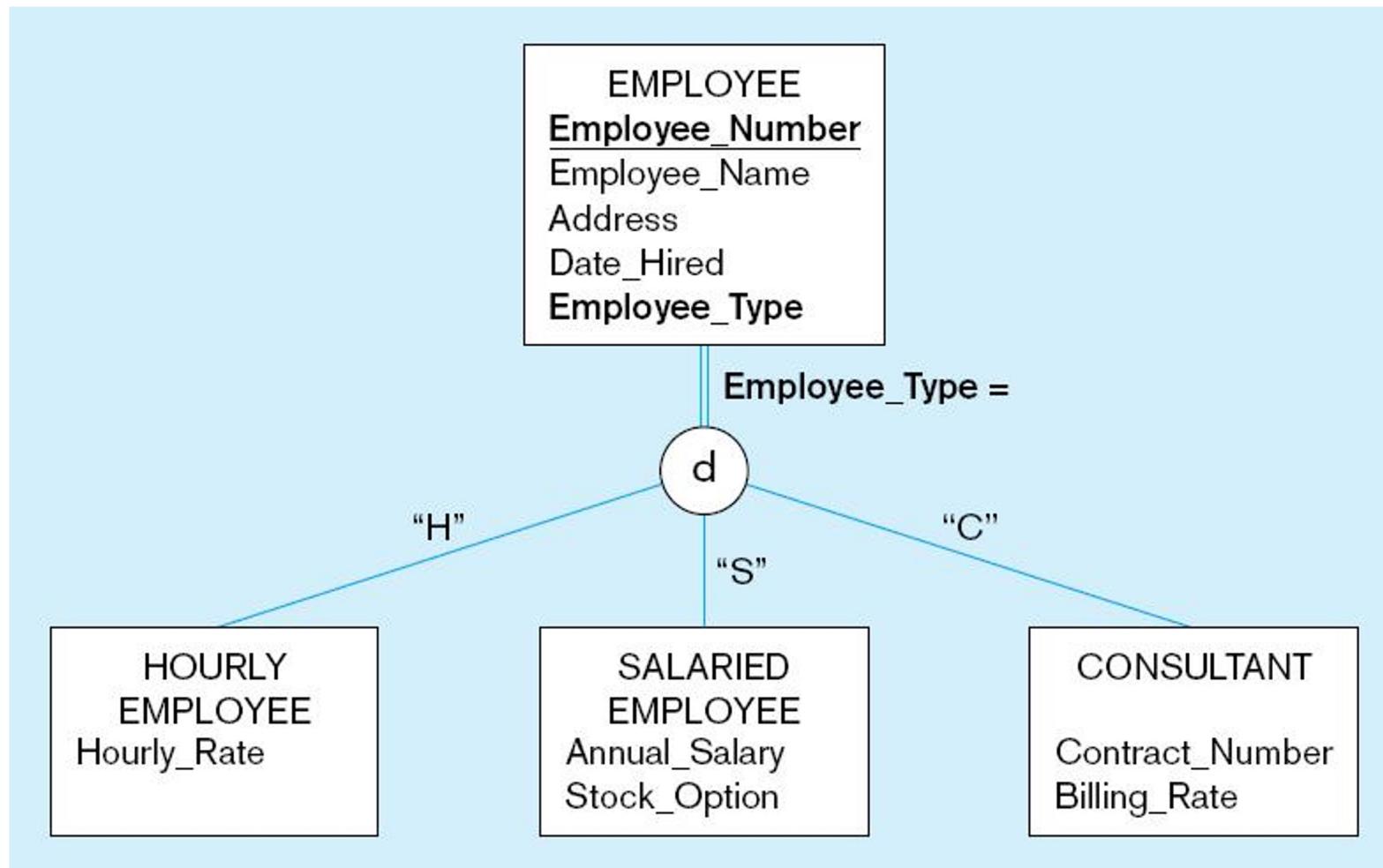
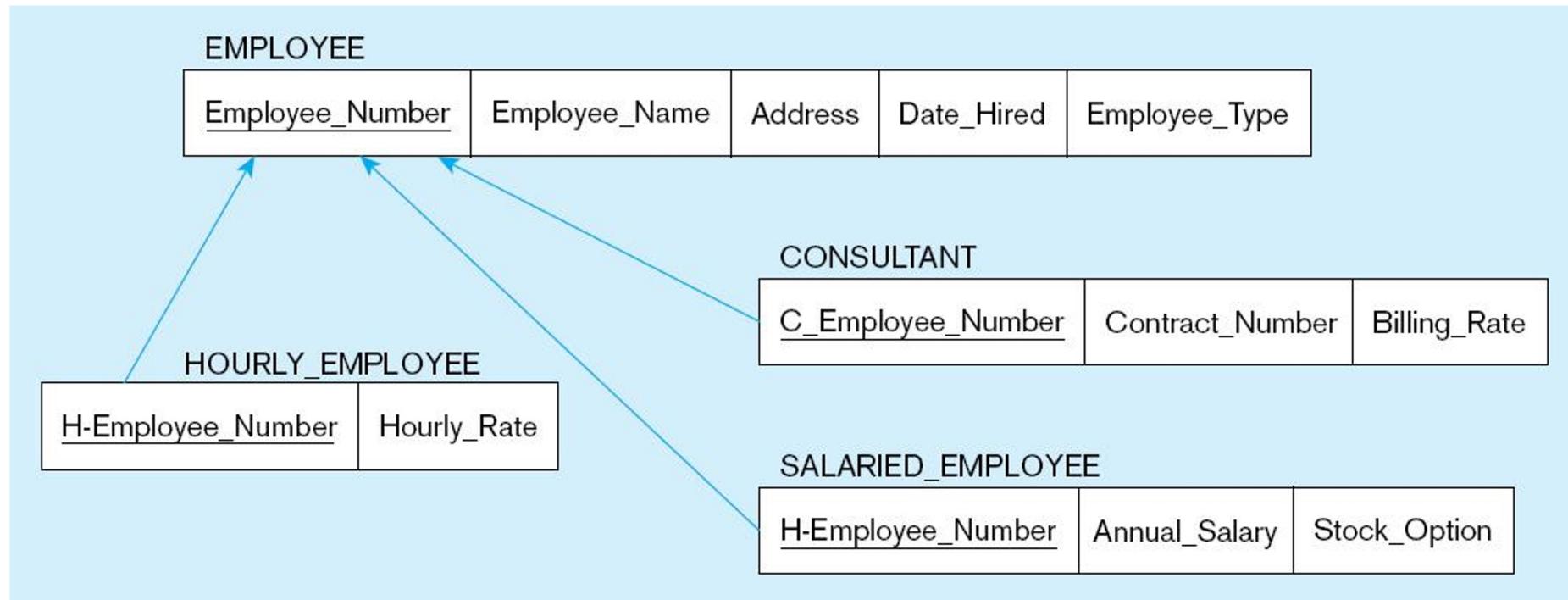


Figure 5-21

## Mapping Supertype/subtype relationships to relations



These are implemented as one-to-one  
relationships

# Data Normalization

- Primarily a tool to validate and improve a logical design so that it satisfies certain constraints that ***avoid unnecessary duplication of data***
- The process of decomposing relations with anomalies to produce smaller, ***well-structured*** relations

# Well-Structured Relations

- A relation that contains minimal data redundancy and allows users to insert, delete, and update rows without causing data inconsistencies
- Goal is to avoid anomalies
  - **Insertion Anomaly**—adding new rows forces user to create duplicate data
  - **Deletion Anomaly**—deleting rows may cause a loss of data that would be needed for other future rows
  - **Modification Anomaly**—changing data in a row forces changes to other rows because of duplication

**General rule of thumb: A table should not pertain to more than one entity type**

# Example–Figure 5-2b

EMPLOYEE2					
Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/200X
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/200X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/200X
110	Chris Lucero	Info Systems	43,000	C++	4/22/200X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/200X
150	Susan Martin	Marketing	42,000	Java	8/12/200X

Question–Is this a relation?

Answer–Yes: Unique rows and no multivalued attributes

Question–What's the primary key?

Answer–Composite: Emp\_ID, Course\_Title

# Anomalies in this Table

- **Insertion**—can't enter a new employee without having the employee take a class
- **Deletion**—if we remove employee 140, we lose information about the existence of a Tax Acc class
- **Modification**—giving a salary increase to employee 100 forces us to update multiple records

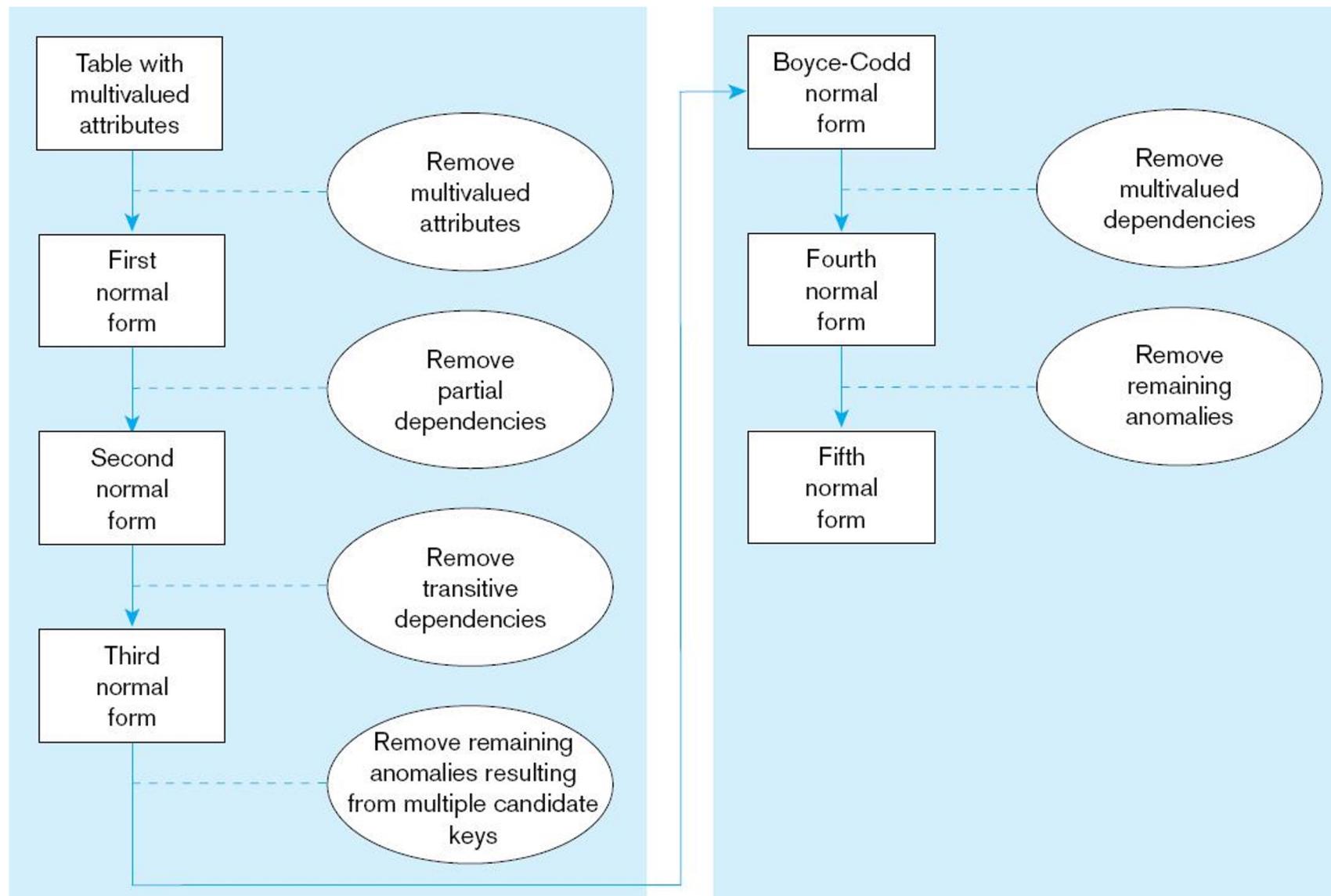
Why do these anomalies exist?

Because there are two themes (entity types) in this one relation. This results in data duplication and an unnecessary dependency between the entities

# Functional Dependencies and Keys

- Functional Dependency: The value of one attribute (the ***determinant***) determines the value of another attribute
- Candidate Key:
  - A unique identifier. One of the candidate keys will become the primary key
    - E.g. perhaps there is both credit card number and SS# in a table...in this case both are candidate keys
  - Each non-key field is functionally dependent on every candidate key

## Figure 5.22 Steps in normalization



# First Normal Form

- No multivalued attributes
- Every attribute value is atomic
- Fig. 5-25 *is not* in 1<sup>st</sup> Normal Form (multivalued attributes) □ it is not a relation
- Fig. 5-26 *is* in 1<sup>st</sup> Normal form
- ***All relations are in 1<sup>st</sup> Normal Form***

# Table with multivalued attributes, not in 1<sup>st</sup> normal form

**Figure 5-25** INVOICE date (Pine Valley Furniture Company)

Order_ID	Order_Date	Customer_ID	Customer_Name	Customer_Address	Product_ID	Product_Description	Product_Finish	Unit_Price	Ordered_Quantity
1006	10/24/2008	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
					5	Writer's Desk	Cherry	325.00	2
					4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2008	6	Furniture Gallery	Boulder, CO	11	4-Dr Dresser	Oak	500.00	4
					4	Entertainment Center	Natural Maple	650.00	3

Note: this is NOT a relation

# Table with no multivalued attributes and unique rows, in 1<sup>st</sup> normal form

**Figure 5-26** INVOICE relation (1NF) (Pine Valley Furniture Company)

Order_ID	Order_Date	Customer_ID	Customer_Name	Customer_Address	Product_ID	Product_Description	Product_Finish	Unit_Price	Ordered_Quantity
1006	10/24/2008	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
1006	10/24/2008	2	Value Furniture	Plano, TX	5	Writer's Desk	Cherry	325.00	2
1006	10/24/2008	2	Value Furniture	Plano, TX	4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2008	6	Furniture Gallery	Boulder, CO	11	4-Dr Dresser	Oak	500.00	4
1007	10/25/2008	6	Furniture Gallery	Boulder, CO	4	Entertainment Center	Natural Maple	650.00	3

Note: this is a relation, but not a well-structured one

# Anomalies in this Table

- **Insertion**—if new product is ordered for order 1007 of existing customer, customer data must be re-entered, causing duplication
- **Deletion**—if we delete the Dining Table from Order 1006, we lose information concerning this item's finish and price
- **Update**—changing the price of product ID 4 requires update in several records

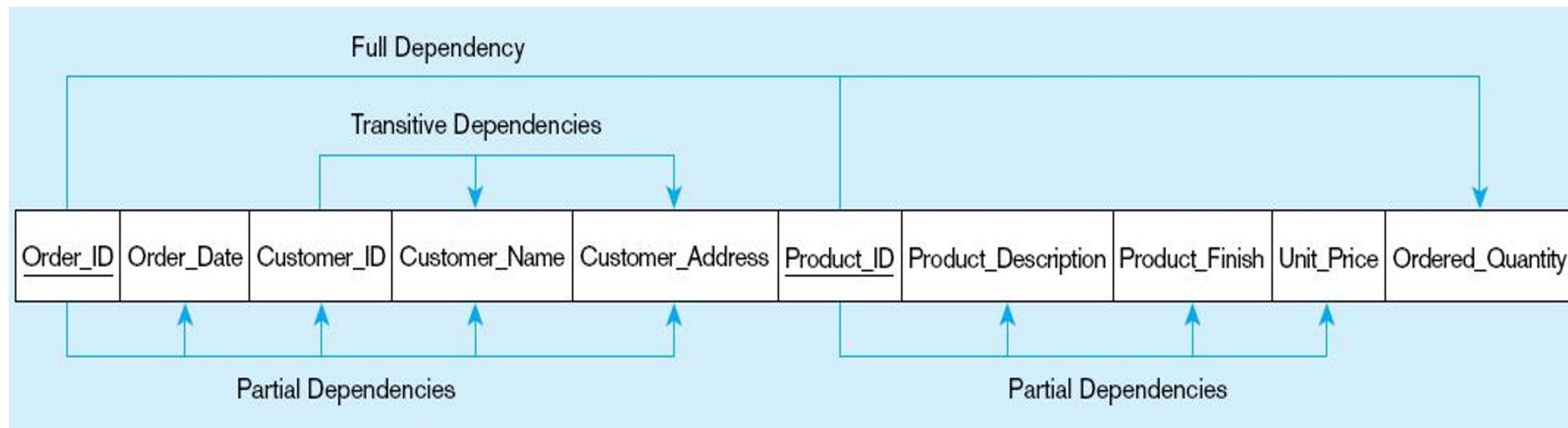
Why do these anomalies exist?

Because there are multiple themes (entity types) in one relation. This results in duplication and an unnecessary dependency between the entities

# Second Normal Form

- 1NF PLUS ***every non-key attribute is fully functionally dependent on the ENTIRE primary key***
  - Every non-key attribute must be defined by the entire key, not by only part of the key
  - No partial functional dependencies

## Figure 5-27 Functional dependency diagram for INVOICE



**Order\_ID ↳ Order\_Date, Customer\_ID, Customer\_Name, Customer\_Address**

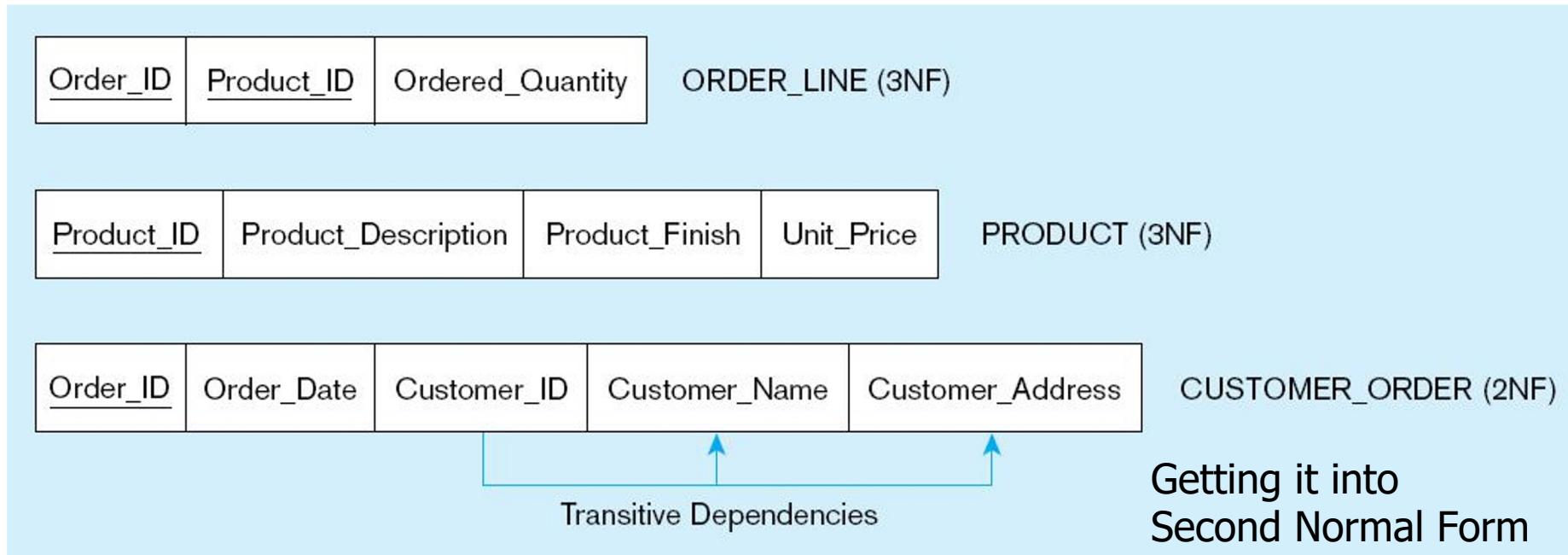
**Customer\_ID ↳ Customer\_Name, Customer\_Address**

**Product\_ID ↳ Product\_Description, Product\_Finish, Unit\_Price**

**Order\_ID, Product\_ID ↳ Order\_Quantity**

**Therefore, NOT in 2<sup>nd</sup> Normal Form**

## Figure 5-28 Removing partial dependencies

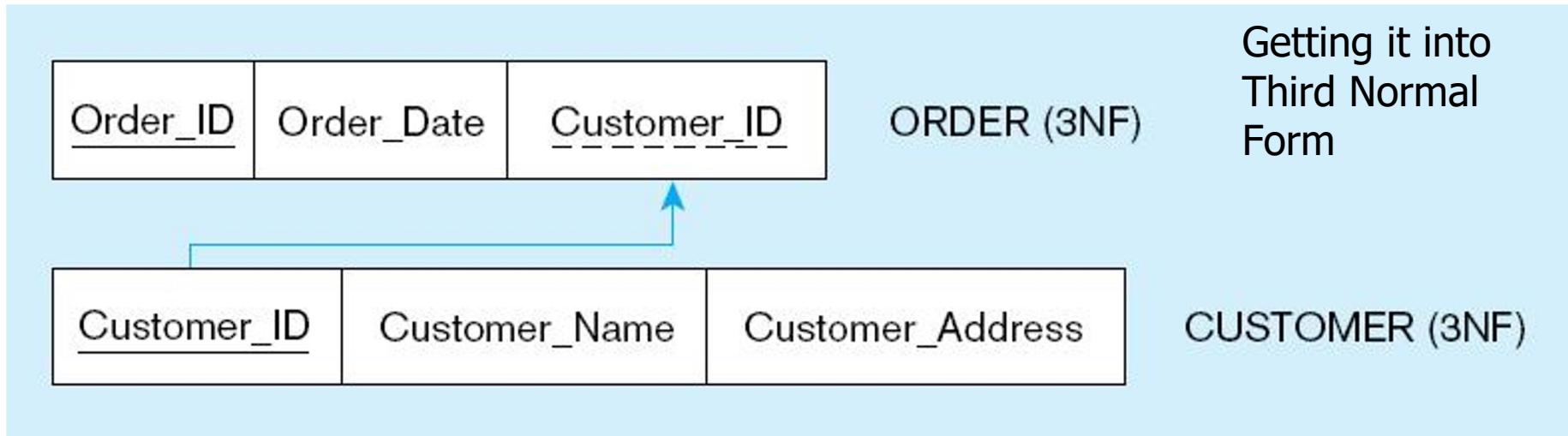


Partial dependencies are removed, but there are still transitive dependencies

# Third Normal Form

- 2NF PLUS ***no transitive dependencies*** (functional dependencies on non-primary-key attributes)
- Note: This is called transitive, because the primary key is a determinant for another attribute, which in turn is a determinant for a third
- Solution: Non-key determinant with transitive dependencies go into a new table; non-key determinant becomes primary key in the new table and stays as foreign key in the old table

## Figure 5-29 Removing partial dependencies



Transitive dependencies are removed