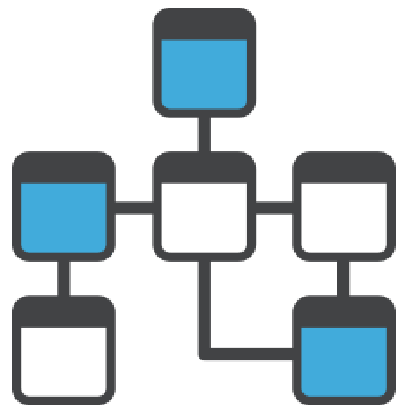
Read Chapters 3, 4 and 14

# Database Modeling




**Dr. Abdelkarim Erradi**

Computer & Engineering Science Dept.

**QU**

# Outline

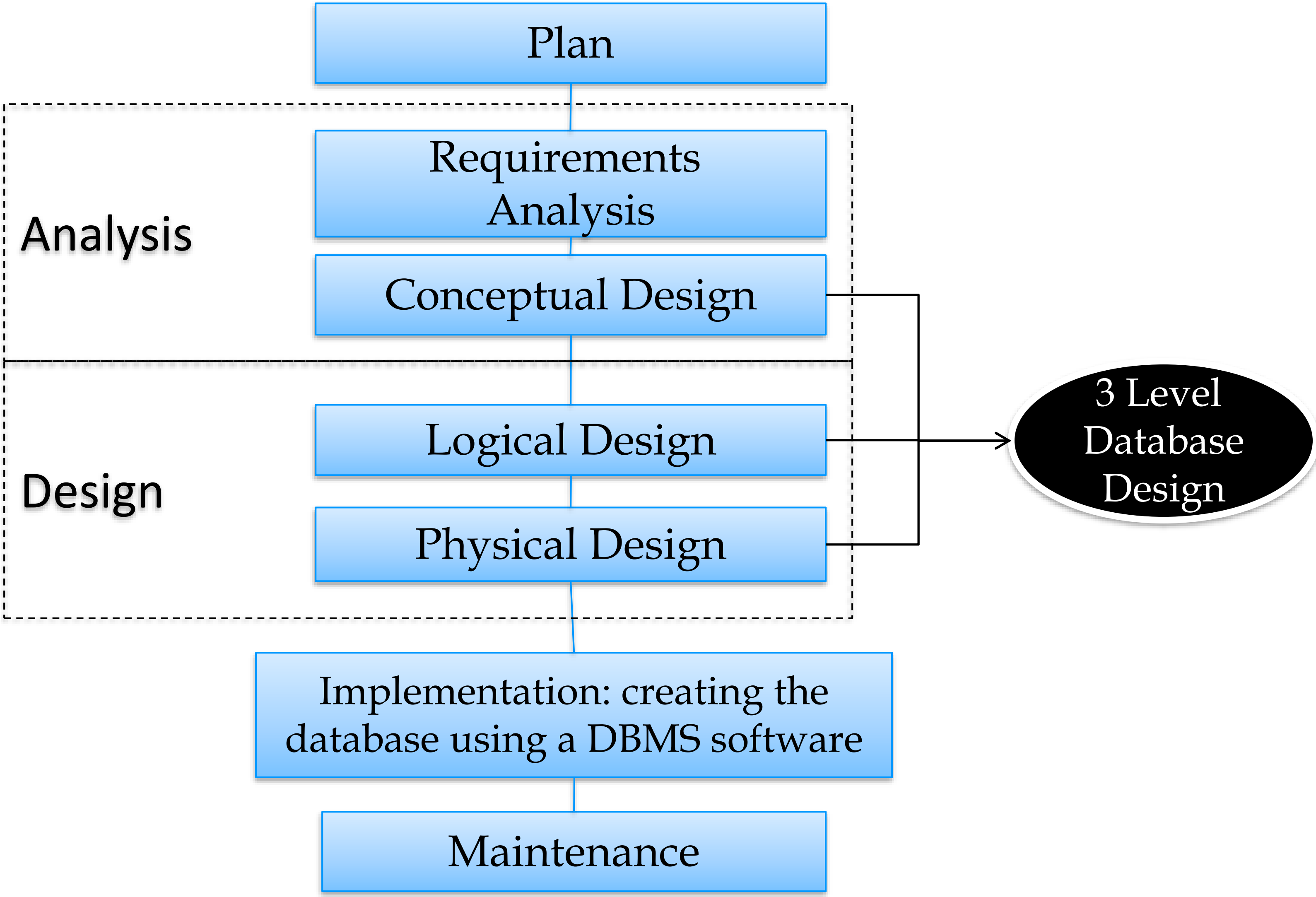
- Conceptual Data Modeling
- Extended Entity-Relationship (EER) model
- Mapping ER Diagram to Logical Model
- Normalization

# Conceptual Data Modeling using Entity-Relationship Diagram (ERD)

# Data Model

- A **data model** is collection of concepts for describing the data in a database
  – defines only the logical model, and NOT a physical storage of the data (i.e., how data is stored)
- Contains:
  – **Structure (schema)**: The definition of entities, their attributes and relationships
  – **Integrity Constraints**: Ensure the database's contents satisfy constraints.
- Many models exist:
  – **Relational**
  – Key/Value
  – Graph
  – Document
  – Column-family
  – Array / Matrix

# Database Development




→ Similar to software development

# Database Design Steps in Practice

- Requirements Analysis
  - User needs; what must database do?
- Conceptual Design
  - Database schema in terms of entities and relationships
- Logical Design
  - Add attributes to entities and relationships
  - Schema Refinement: Normalization to reduce redundancy
  - Translate the ER into a particular RDBMS DB schema
- Physical Design - indexes, physical data organization
- Security Design - who accesses what

# Conceptual Database Design

- Use the Entity Relationship (ER) model to develop a high level description of the data
- Identify the entities and relationships in the problem domain
- Identify what information about these entities and relationships is to be stored in the database
- Draw an *Entity Relationship* diagram (ERD)
- Identify the integrity constraints that apply to the entities and relationships
- Check with the client that the ER model that has been developed is correct

# Logical Database Design

- Determine which data model should be used to implement the database (e.g., relational model)
- Determine which DBMS to use
  - In most cases this means deciding which existing DBMS product to purchase or license
- Map, or translate, the conceptual schema to a database schema of the chosen model
- There are two major problems to be avoided
  - ***Redundancy*** – information should not be repeated
  - ***Incompleteness*** – it should be possible to record all the desired information

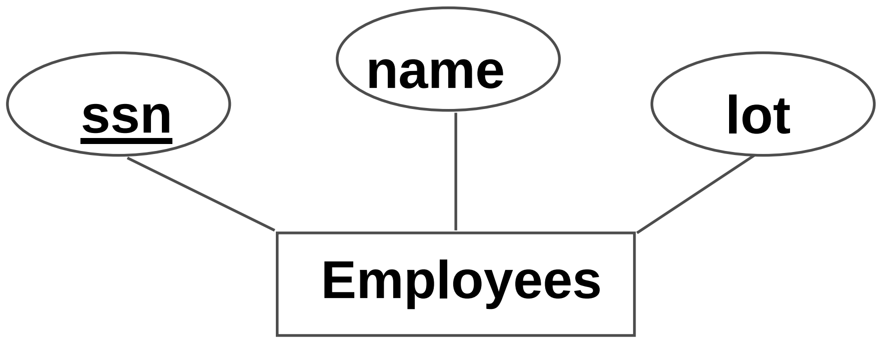
# The Entity-Relationship Model

- The most common conceptual data model
- The major components of the ER Model are:
  - **Entity** – something in the real world that we wish to track and store data about (employee, item)
  - **Attribute** – A characteristic of an entity or relationship (EmployeeID, Item Description)
  - **Relationship** – A link that connect between entities (e.g. A customer (*entity*) buys (*relationship*) a product (*entity*)
  - **Constraints** which restrict relationships, e.g. an account *must be* owned by a customer

# Entity-Relationship Diagram

- Entity-relationship diagram show the structure of a database graphically
  - Simple symbols: rectangles, diamonds, ovals and lines represent the components of the ER model
- Dr. Peter Chen developed ERD. The ERD notation become a popular tool for relational database design
- There are many variations of ER diagrams
  - So don't expect the symbols in every ER diagram you see to be exactly the same!
  - Some common variations are discussed in this presentation

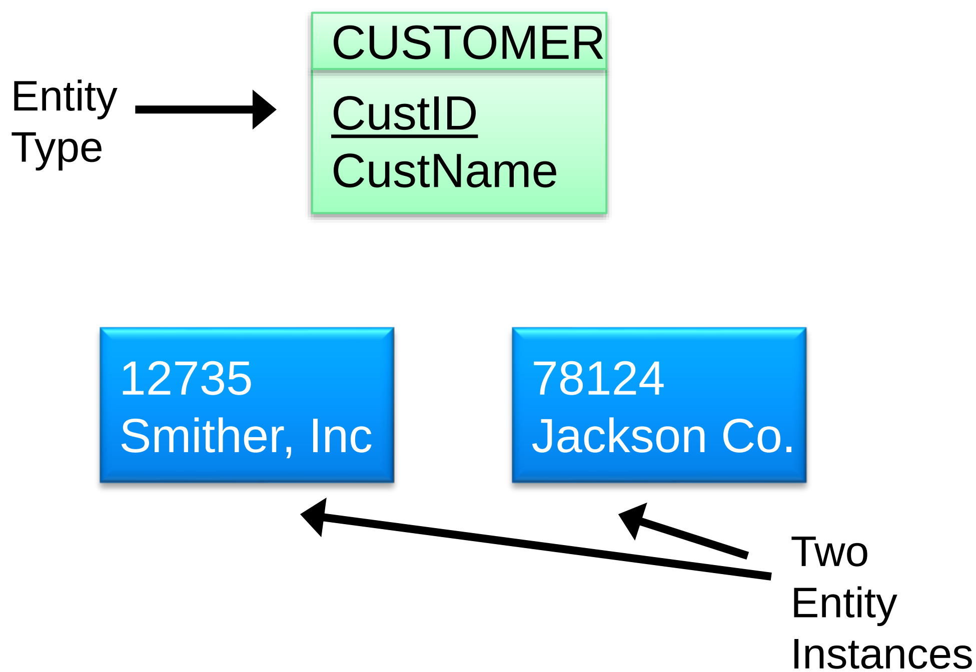
# Entities and Entity Sets




- *Entity:* Real-world object, distinguishable from other objects. An entity is described using a set of *attributes*.
- *Entity Type (or Entity Set)*: A collection of similar entities. E.g., all employees.
  - Each entity set has a *key (underlined)*.
  - A *key* is a set of attributes whose values *uniquely* identify an entity in an entity set

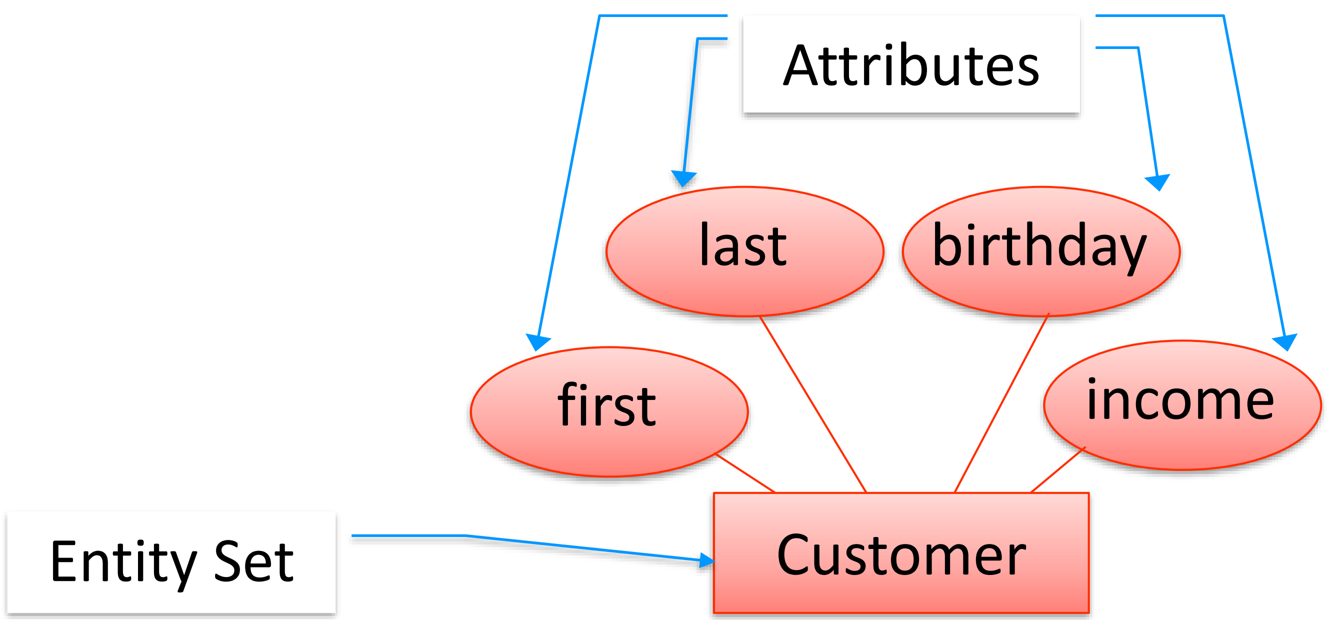
# Instance versus Type

- An *entity type* is a description of the structure and format of the instance of the entity
- An *entity instance* is a specific occurrence of an entity type



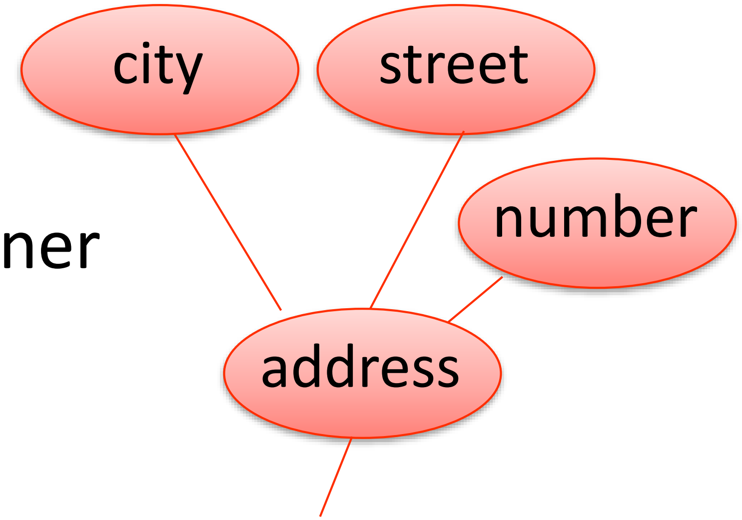
# Attributes

- Attributes describe the properties of entities or relationships.
  – Each attribute has a domain: a set of all possible attribute values
  – Correspond to fields or columns of a table.
  – Avoid derived attribute, e.g., age



# Composite Attributes

- Composite attributes are divided into subparts
  - e.g. address is composed of city, street and number
- They group related attributes together to make the model cleaner
- Some versions of the ER model disallow composite attributes
  - Replacing them with their subparts alone

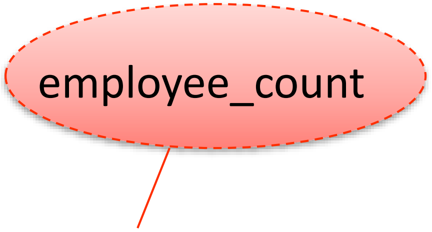


# Multivalued Attributes

- A multivalued attribute is a set of values
  - All of the same type
  - e.g. phone numbers
- Some versions of the ER model disallow multivalued attributes
  - Replacing them with another entity

# Derived Attributes

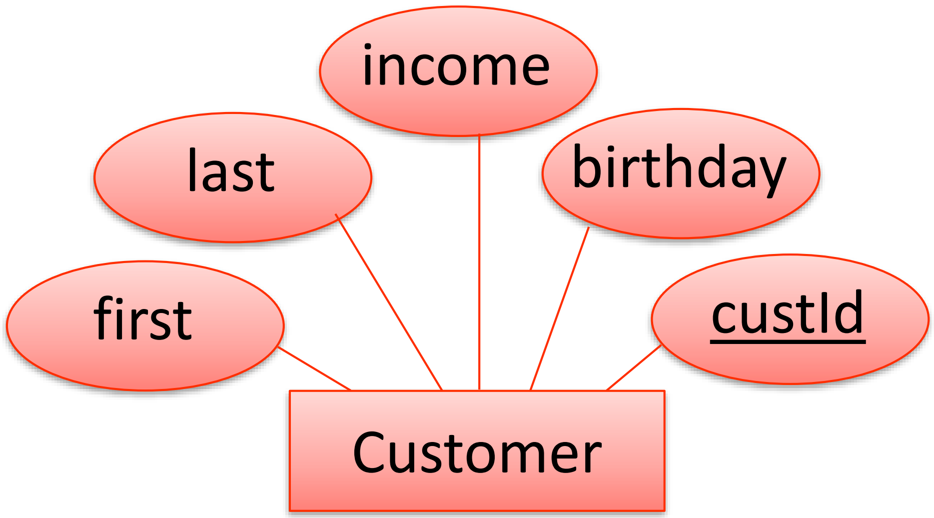
- The value of a derived attribute can be derived from other values
  - Belonging to related attributes or entities
  - e.g. employee_count for a department
    - Calculated by counting the number of employees in the department
- Derived attributes do not need to be stored in the database
  - They can be calculated when required



# Selecting a Primary Key

- A *key* is a set of attributes whose values *uniquely* identify an entity in an entity set
- The primary key should be chosen so that its attributes never (or very rarely) change
  – Including an address as part of a primary key is therefore not recommended
  – E.g., Qatari National ID make a good primary key
- It is sometimes useful to *generate* a unique primary key for entities

# Primary Key



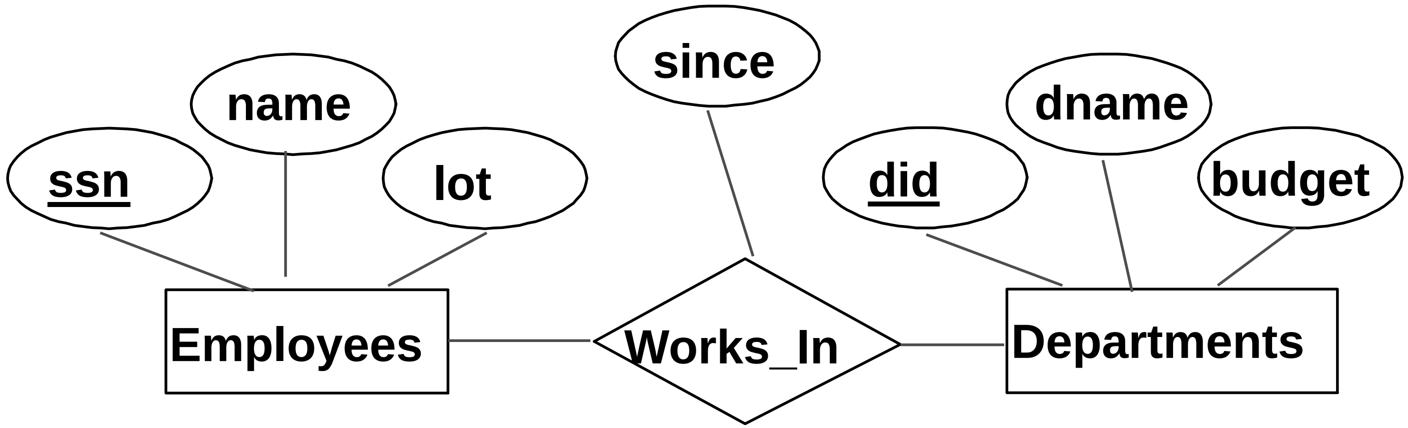

Candidate Key 1: {first, last, birthday}*

Candidate Key 2: {custId}

Primary Key: {custId}

*assuming (unrealistically) that there are no two people with the same first name, last name and birthday

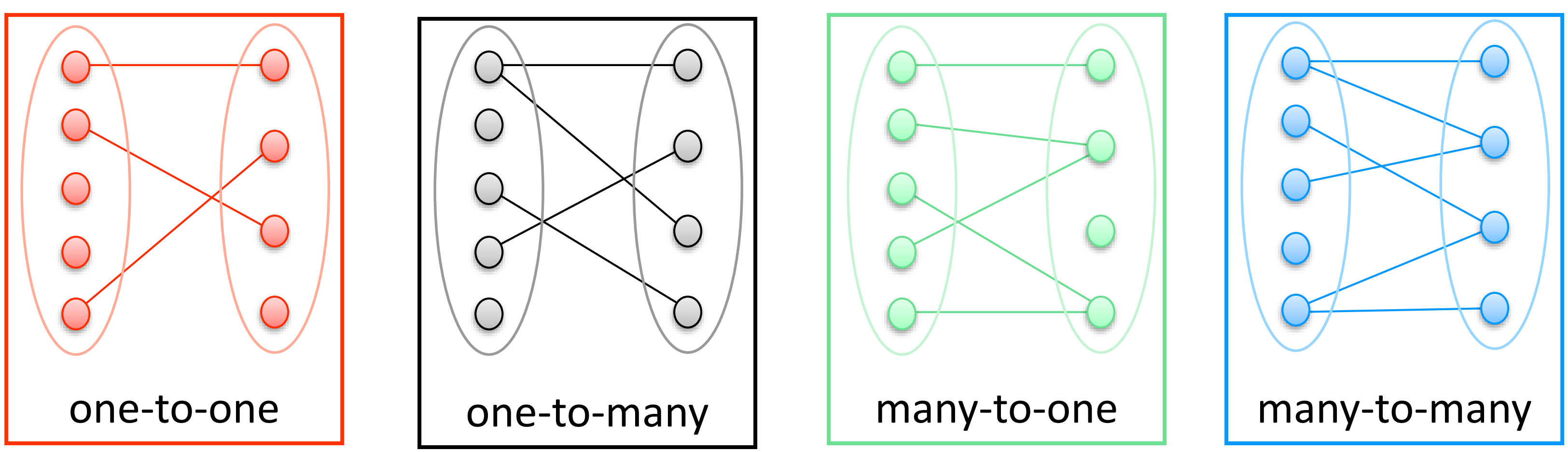
# Relationship




- *Relationship*: Association among two or more entities. E.g., Ali works in Pharmacy department.
  - relationships can have their own attributes.
- *Relationship Type (or Relationship Set)*: Collection of similar relationships.

# Cardinalities

- Express the number of entities to which another entity can be associated via a relationship set.
  - One-to-one (1:1)
  - One-to-many (1:N)
  - Many-to-many (M:N)

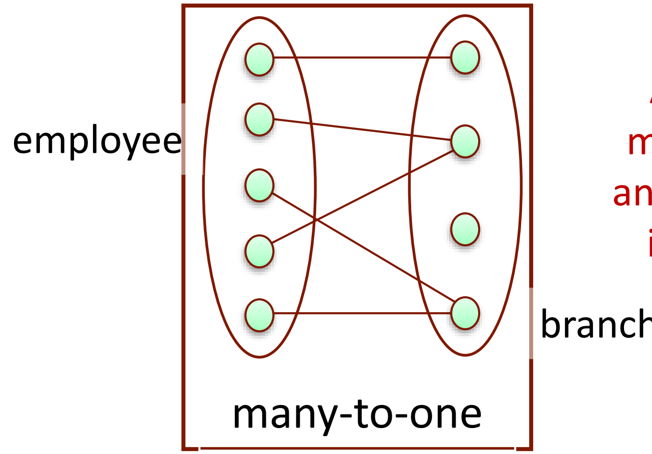


# Cardinalities

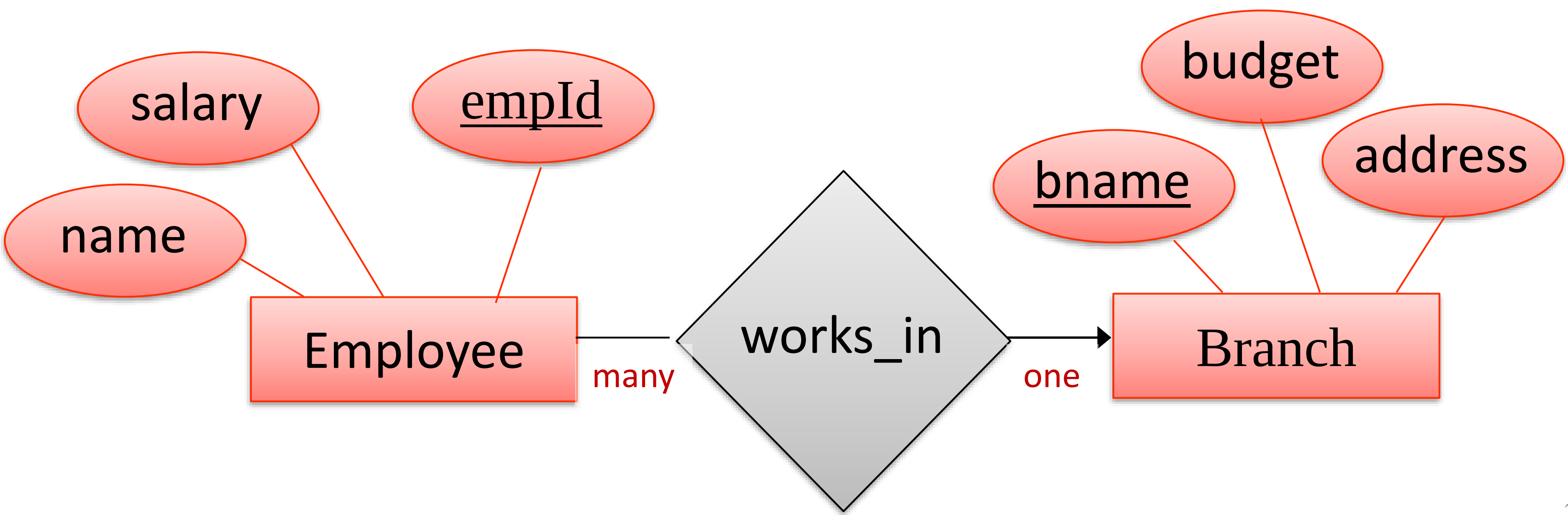
- Assume entity sets A and B
- One-to-one
  - An entity in A associates with *at most* one entity in B, an entity in B associates with *at most* one entity in A
- One-to-many (A to B)
  - An entity in A associates with *any number* of entities in B, an entity in B associates with at most one entity in A
- Many-to-many
  - An entity in A associates with *any number* of entities in B, an entity in B associates with *any number* of entities in A

# Cardinalities Example

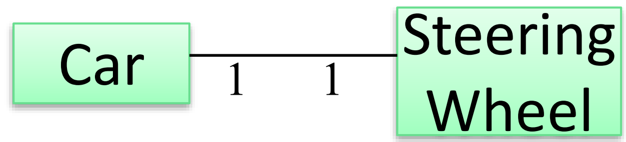
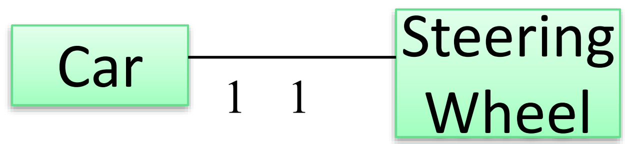
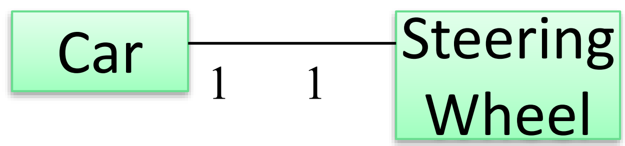
- An employee can work in only one branch
- A branch can have many employees working in it

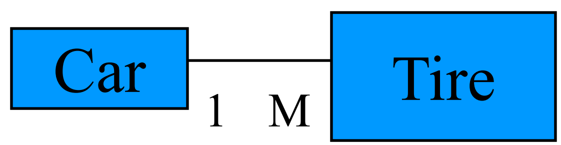
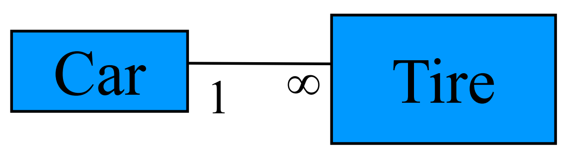
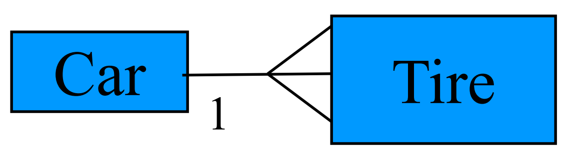



A branch can have many employees but an employee can work in only one branch

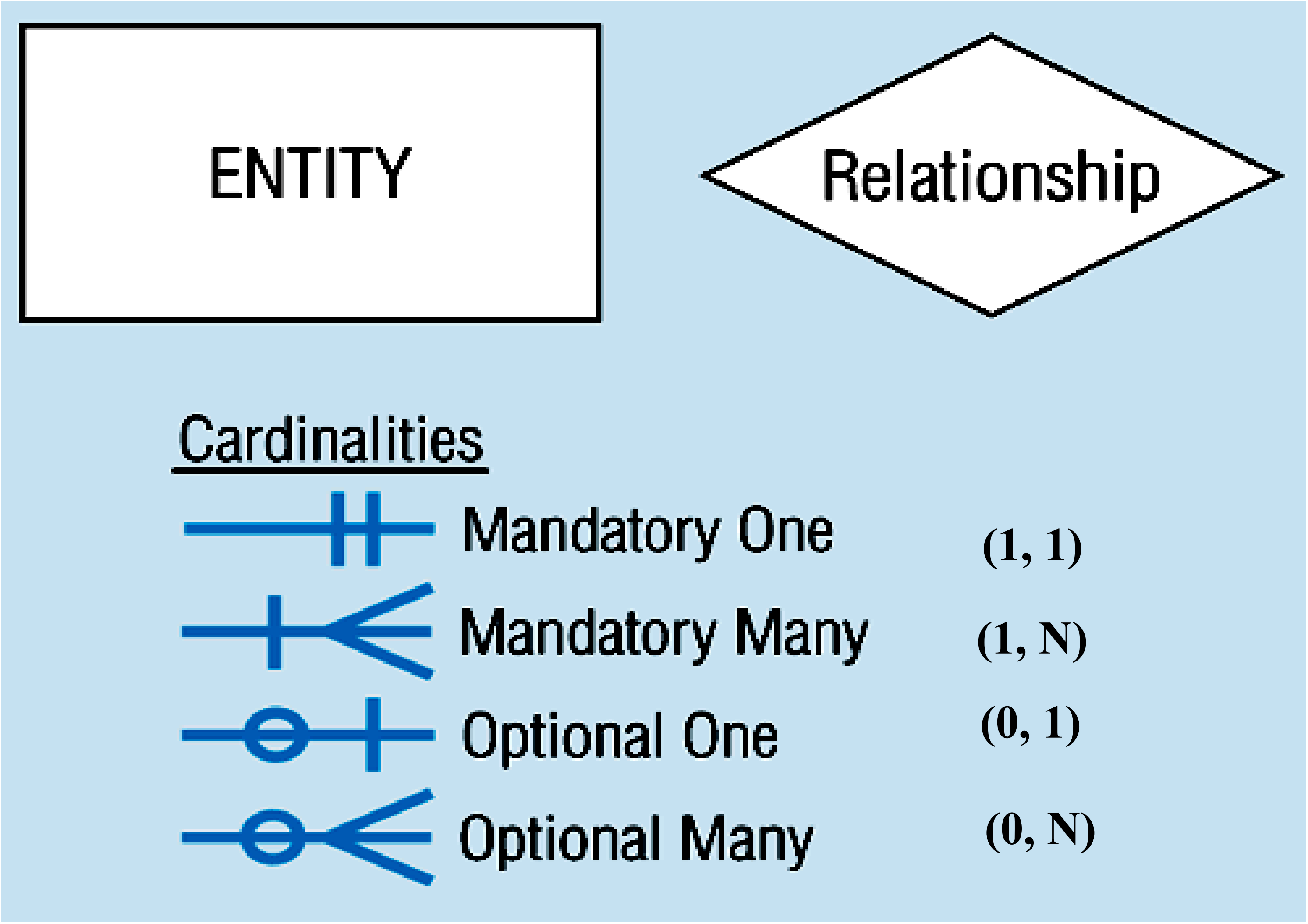


# Example Cardinality Notations

| | |
|---|---|
| Chen |   |
| Infinity |   |
| Crow's Feet |   |

| | |
|---|---|
| Chen |   |
| Infinity |   |
| Crow's Feet |   |

# Chen(min/max) vs. Crows Feet

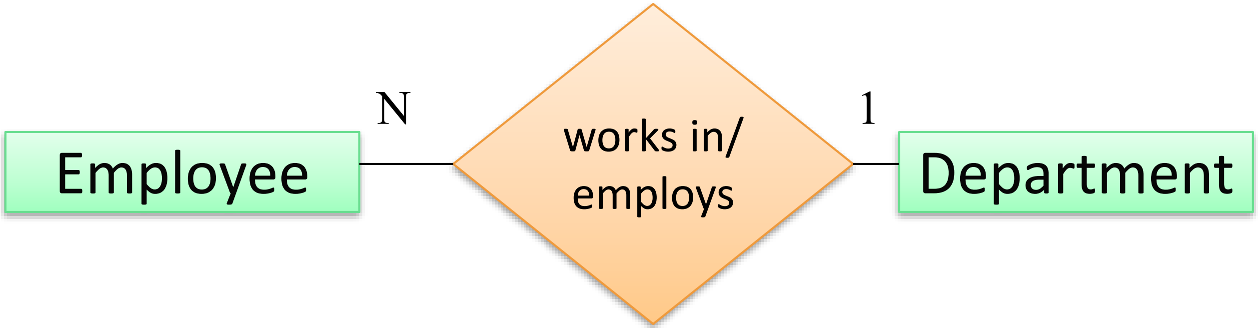


# Examples

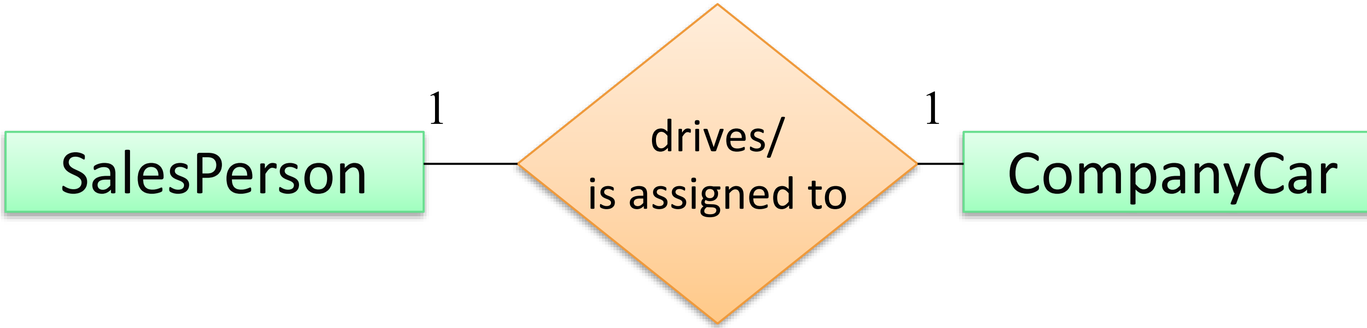
| Relationship | Example | left | right |
|---|---|---|---|
| one-to-one | person ←→ birth certificate | 1 | 1 |
| one-to-one (optional on one side) | person ←→ driving license | 1 | 0..1 |
| many-to-one | person ←→ birth place | 1..* | 1 |
| many-to-many (optional on both sides) | person ←→ book | 0..* | 0..* |
| one-to-many | order ←→ line item | 1 | 1..* |
| many-to-many | course ←→ student | 1..* | 1..* |

# Relationship Examples

**Many-to-One**



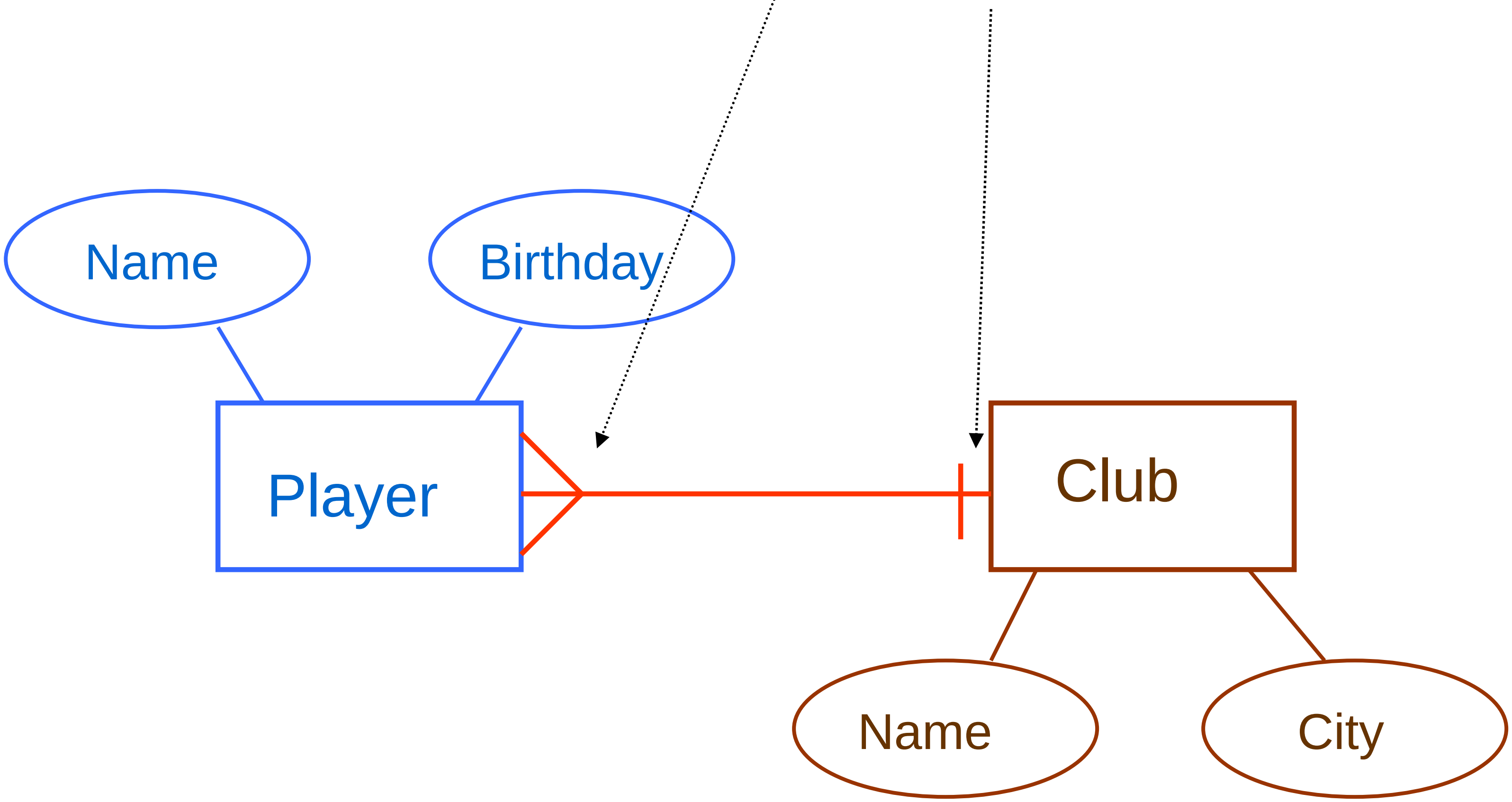

**One-to-One**




**Many-to-Many**

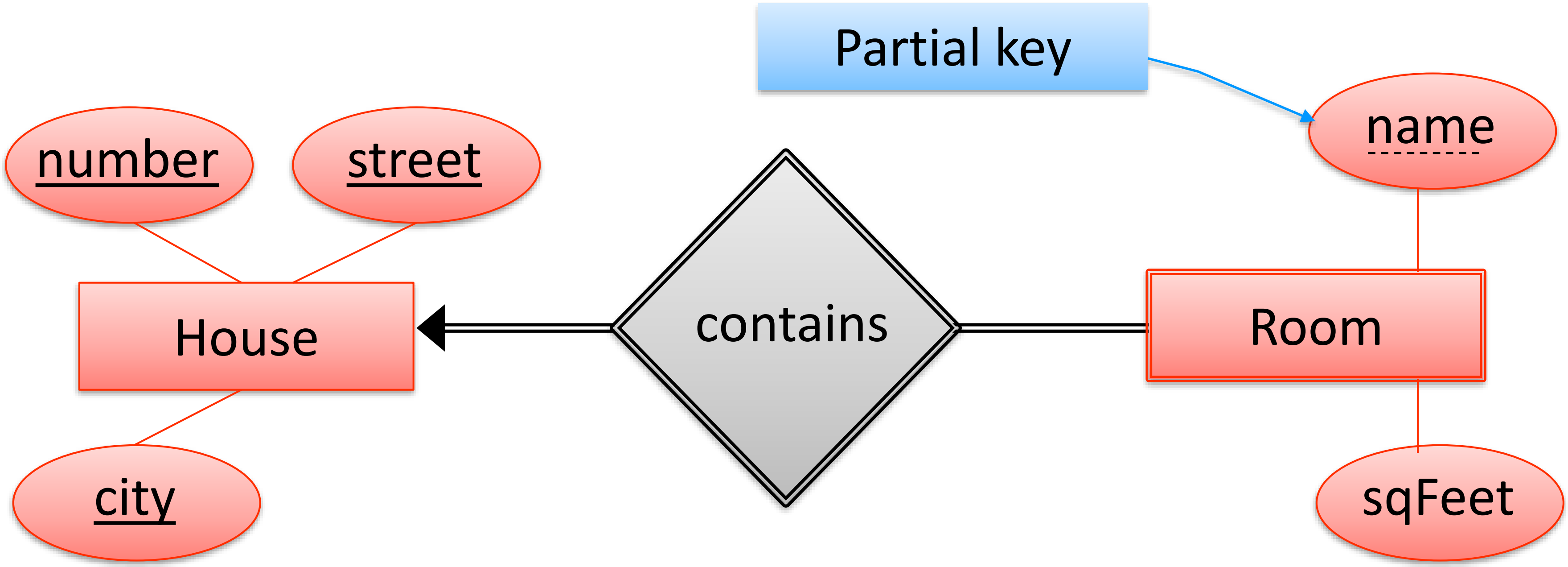
# Relationship Example

- Each club hires many players
- Each player signs with only one club



# Weak Entity Sets

- A weak entity cannot be identified by its own attributes alone
  - A member of a weak entity set is identified by combining its ***partial key*** with the ***primary key*** of the **owner entity set**



# Extended Entity-Relationship (EER) model

# Subclasses

- Similar to inheritance in Object Oriented (OO) modeling
  - Subtype entity inherits all attribute from the supertype entity
  - The subtype entity may also have additional attributes

- Why?
  - Reuse: Common attributes don’t have to be re-defined for each subtype

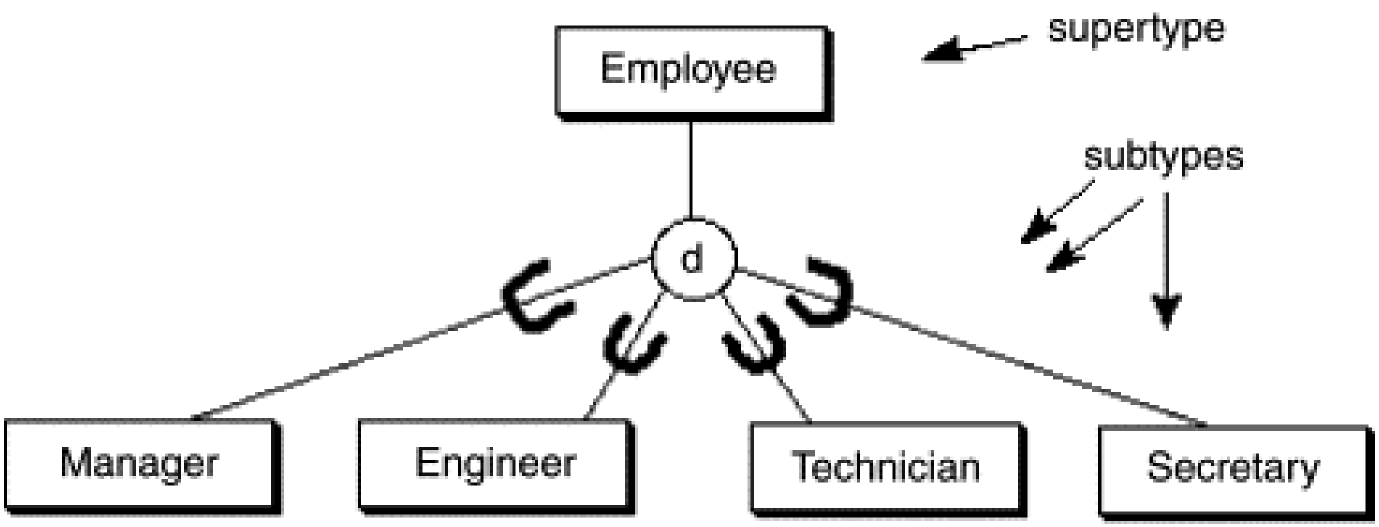
# Designing Subclasses

- The subclass relationship is an ***" is a" relationship***
  - A *"is a"* B (A is the subtype, B the supertype)
  - A *specializes* B or
  - B *generalizes* A
- ***Specialization*** is the process of identifying subsets with additional attributes from an existing entity set
- ***Generalization*** is the process of identifying common attributes of entity sets
  - And creating a new parent entity set with those attributes

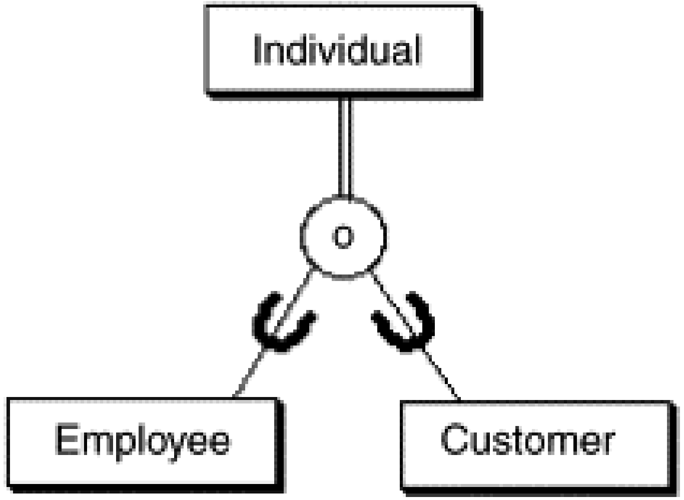
# Disjoint vs. Overlap Constraints

- A *disjoint* constraint requires that a *supertype entity instance* can only belong to one subtype
  - A super class ‘Account' with sub classes 'Saving Account' and 'Current Account'. This is a **disjoint constraint** because a bank account can either be Saving or Current. It cant be both at the same time.
- For an **overlapping** constraint, *the supertype entity instance* can appear in many subtype instance.
  - A super class 'Person' and subclasses 'Customer' and 'Employee'. In this case, a person can be Customer and Employee both. Therefore, overlapping.

# Disjoint vs. Overlap - Example




(a)



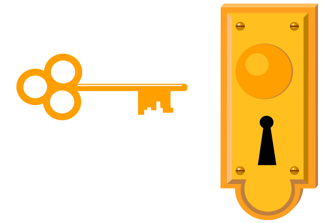

(b)

# Mapping ER Diagram to Logical Model

# Relation

- A database is collection of relations (or tables)
- Definition: A Relation is a **table**
  - Table is made up of rows (records), and columns (attribute or field)
- Characteristics of a Relation:
  - Every relation has a unique name.
  - Every attribute value is atomic (not multivalued, not composite)
  - Every row is unique (can’t have two rows with exactly the same values for all their fields)
  - Attributes (columns) in a table have unique names
  - The order of the columns is irrelevant
  - The order of the rows is irrelevant

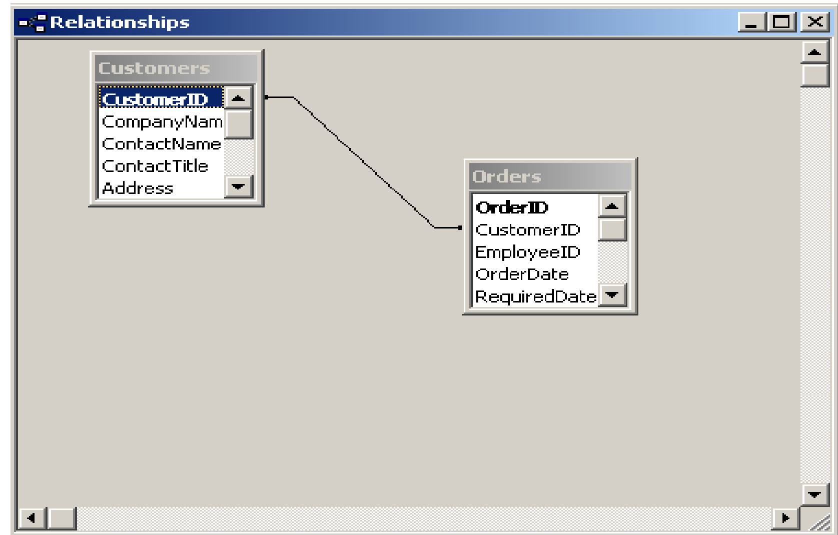
# Key Fields

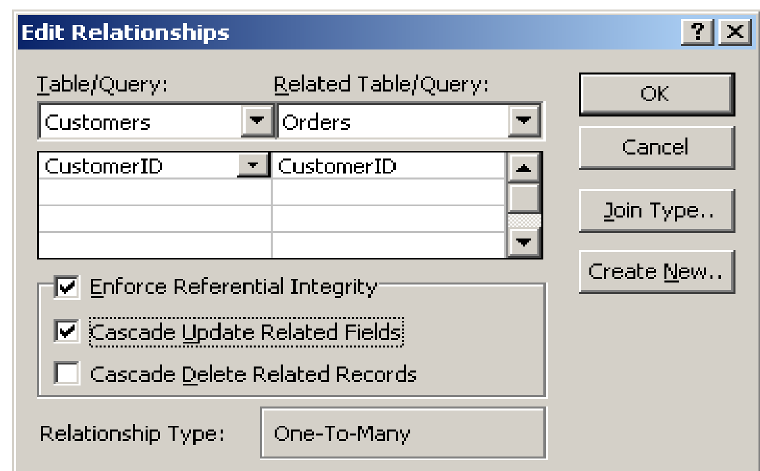


- Keys are special fields that serve two main purposes:
  - ***keys*** are unique identifiers a  record. Examples include employee number, National Card number, etc.
  - ***keys*** are identifiers that enable a dependent relation (on the many side of a relationship) to refer to its parent relation (on the one side of the relationship)
- Keys can be ***simple*** (a single field) or ***composite*** (more than one field)
- ***Surrogate*** key (e.g., auto-incremented primary key)
- Keys usually are used as indexes to speed up the response to user queries

# Referential Integrity

- Rules to preserve relationships
- Prevents orphan records
  - Cannot add record on many side with the related one on the one side
  - Cannot delete from one side if there are related records in the many sides
- But can configure Cascade Update / Cascade Delete



# Example keys

CUSTOMER

| | | | | | |
|---|---|---|---|---|---|
| Customer_ID | Customer_Name | Customer_Address | City * | State * | Postal_Code * |

Primary Key

ORDER

| | | |
|---|---|---|
| Order_ID | Order_Date | Customer_ID |

Foreign Key
(implements 1:N relationship between customer and order)

ORDER LINE

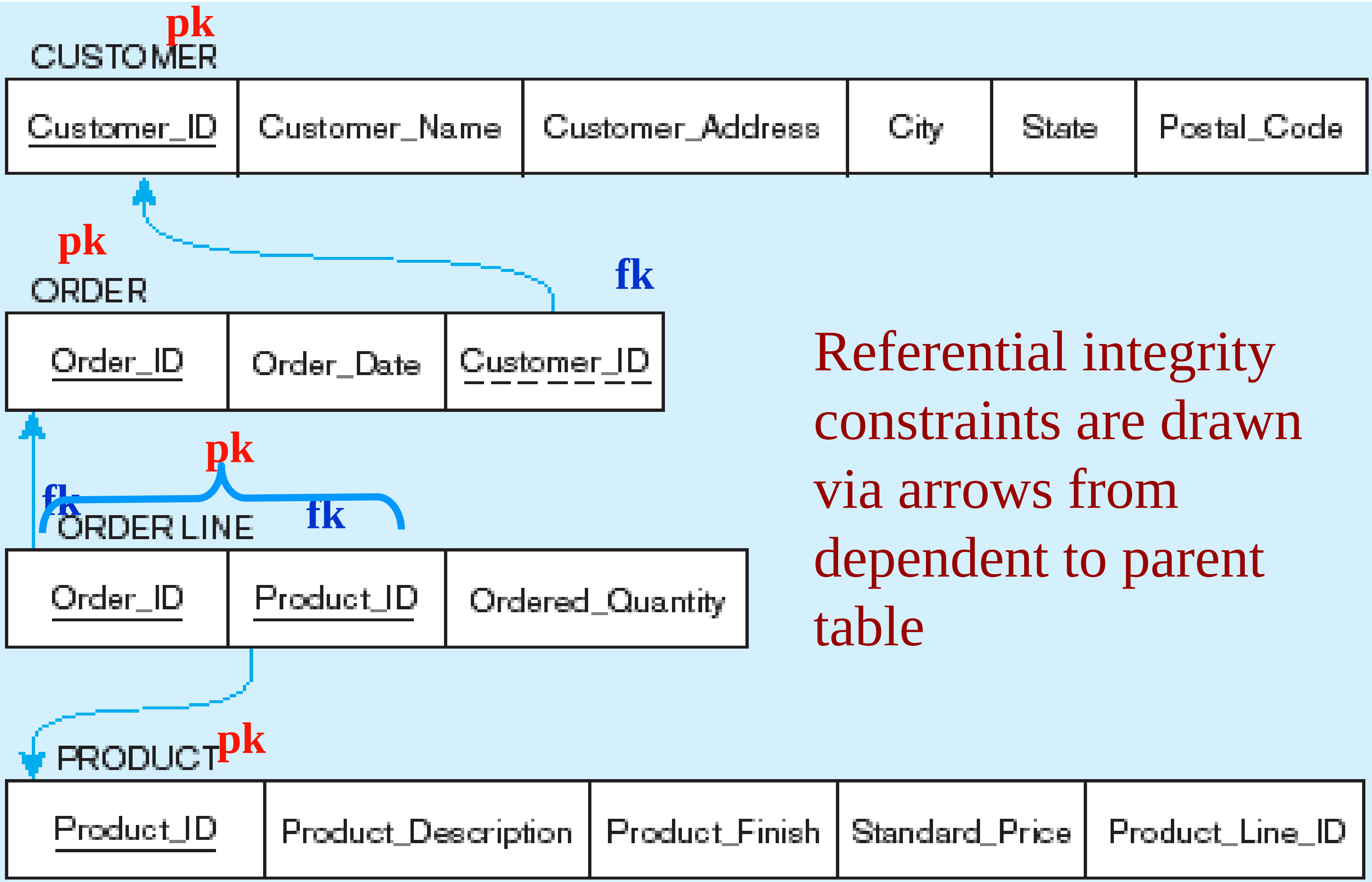
| | | |
|---|---|---|
| Order_ID | Product_ID | Ordered_Quantity |

Combined, these are a *composite primary key* (uniquely identifies the order line)…individually they are *foreign keys* (implement M:N relationship between order and product)

PRODUCT

| | | | | |
|---|---|---|---|---|
| Product_ID | Product_Description | Product_Finish | Standard_Price | Product_Line_ID |

# Referential integrity constraints




Referential integrity constraints are drawn via arrows from dependent to parent table

# Mapping a composite attribute

(a) CUSTOMER entity type with composite attribute

CUSTOMER
Customer_ID
Customer_Name
Customer_Address
(Street, City, State)
Postal_Code

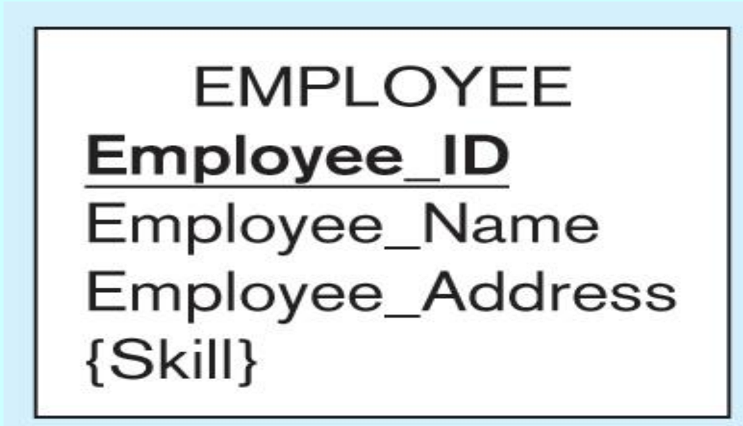
**[Break composite attribute into atomic attributes]**

(b) CUSTOMER relation with address detail

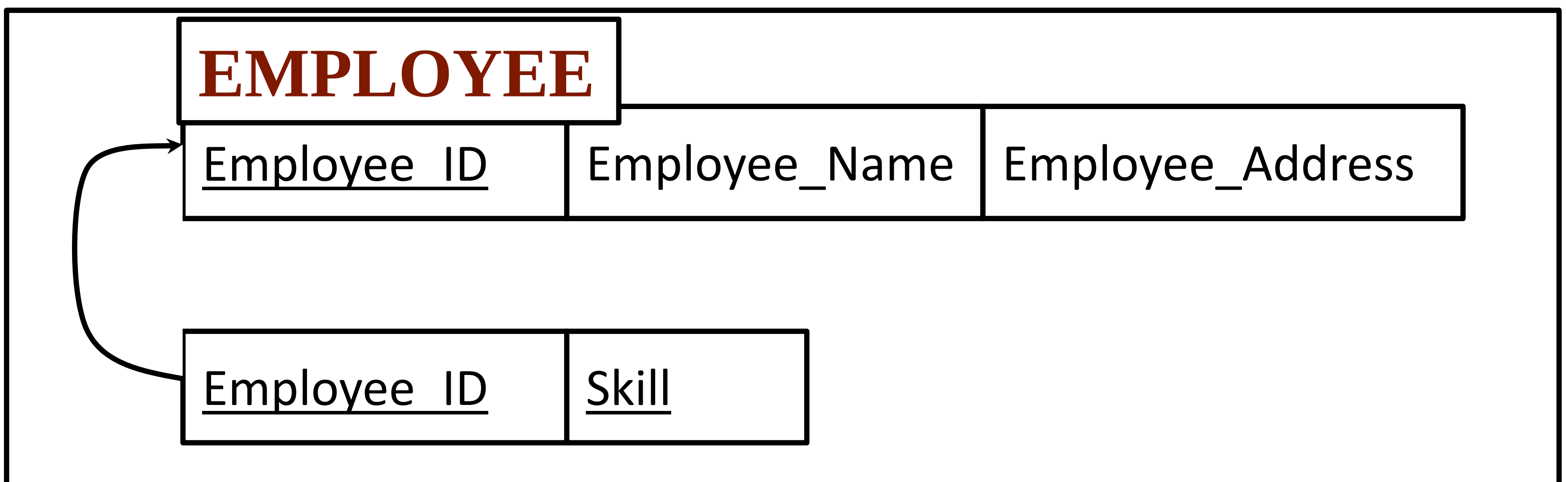
CUSTOMER

| Customer_ID | Customer_Name | Street | City | State | Postal_Code |
| --- | --- | --- | --- | --- | --- |

# Mapping an entity with a multivalued attribute



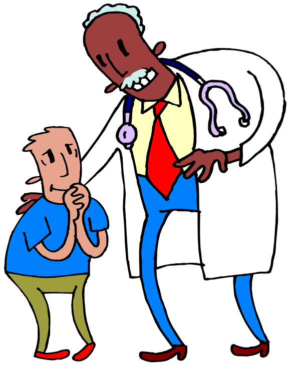

**Multivalued attribute becomes a separate relation with foreign key**




**One–to–many relationship between original entity and new relation**

**[Two relations created with one containing all of the attributes except the multi-valued attribute, and the second one contains the pk (of the first one) and the multi-valued attribute]**

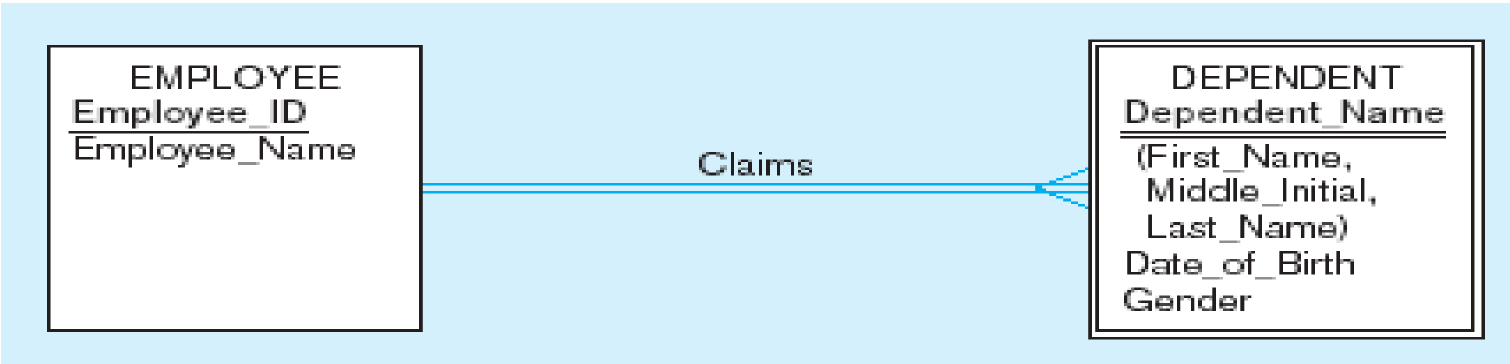
# Mapping Weak Entities



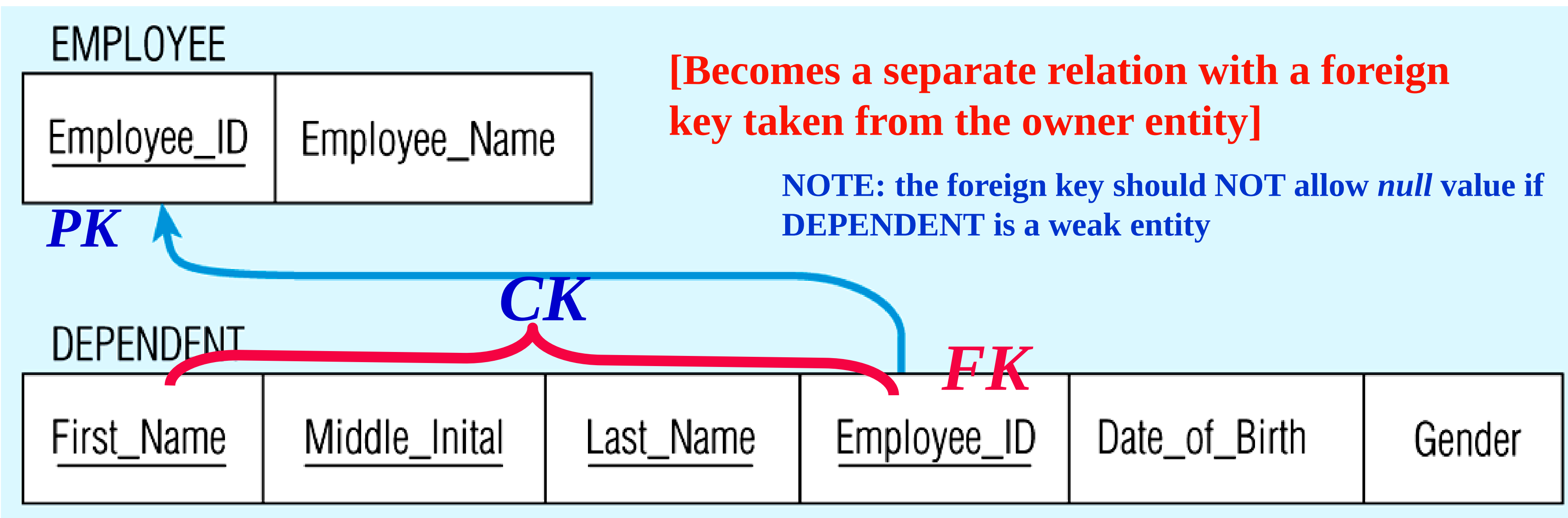
- Week entity cannot exit on its own
- The OrderItem table stores weak entities precisely because an OrderItem has no meaning independent of the Order.
- Becomes a separate relation with a foreign key taken from the owner entity
- Primary key composed of:
    - **Partial identifier** of weak entity
    - **Primary key** of identifying relation (owner entity)

# Example of mapping a weak entity

(a) Weak entity DEPENDENT




(b) Relations resulting from weak entity




**[Becomes a separate relation with a foreign key taken from the owner entity]**

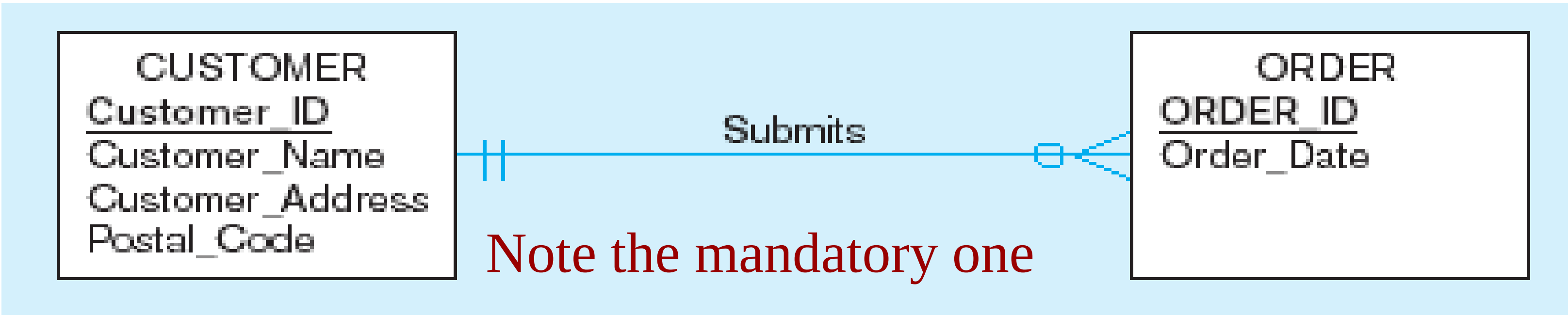
**NOTE: the foreign key should NOT allow *null* value if DEPENDENT is a weak entity**

# Map 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

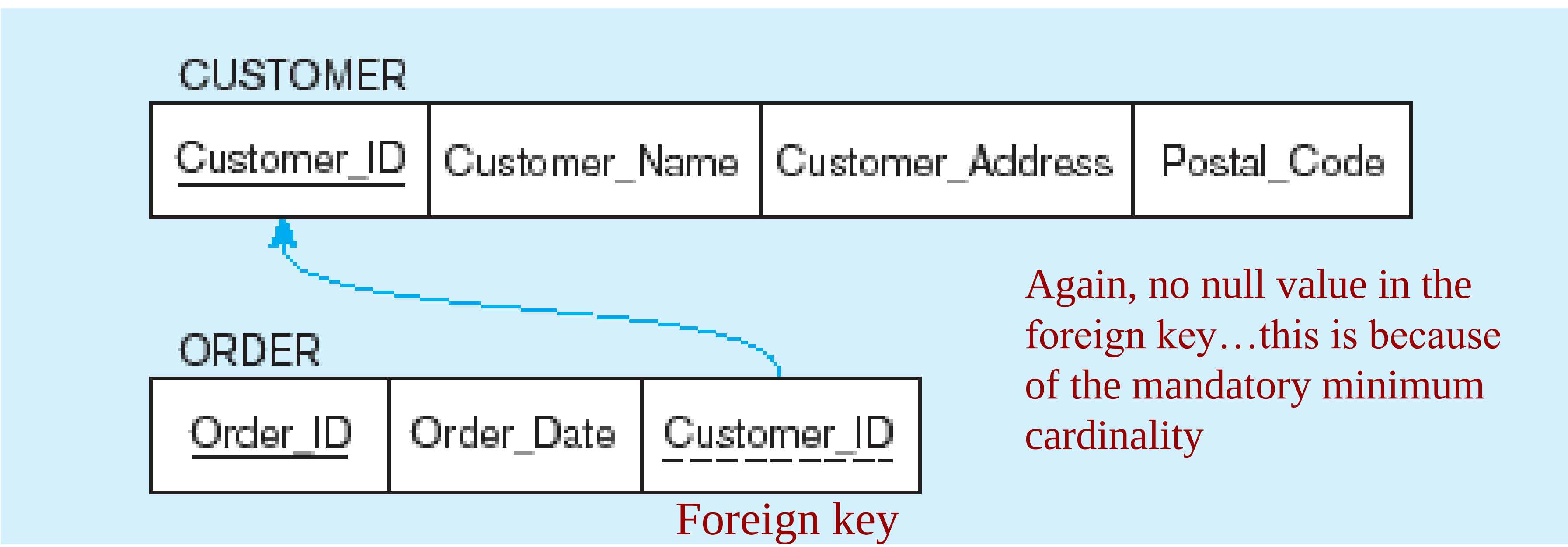
# Mapping a 1:M relationship

(a) Relationship between customers and orders

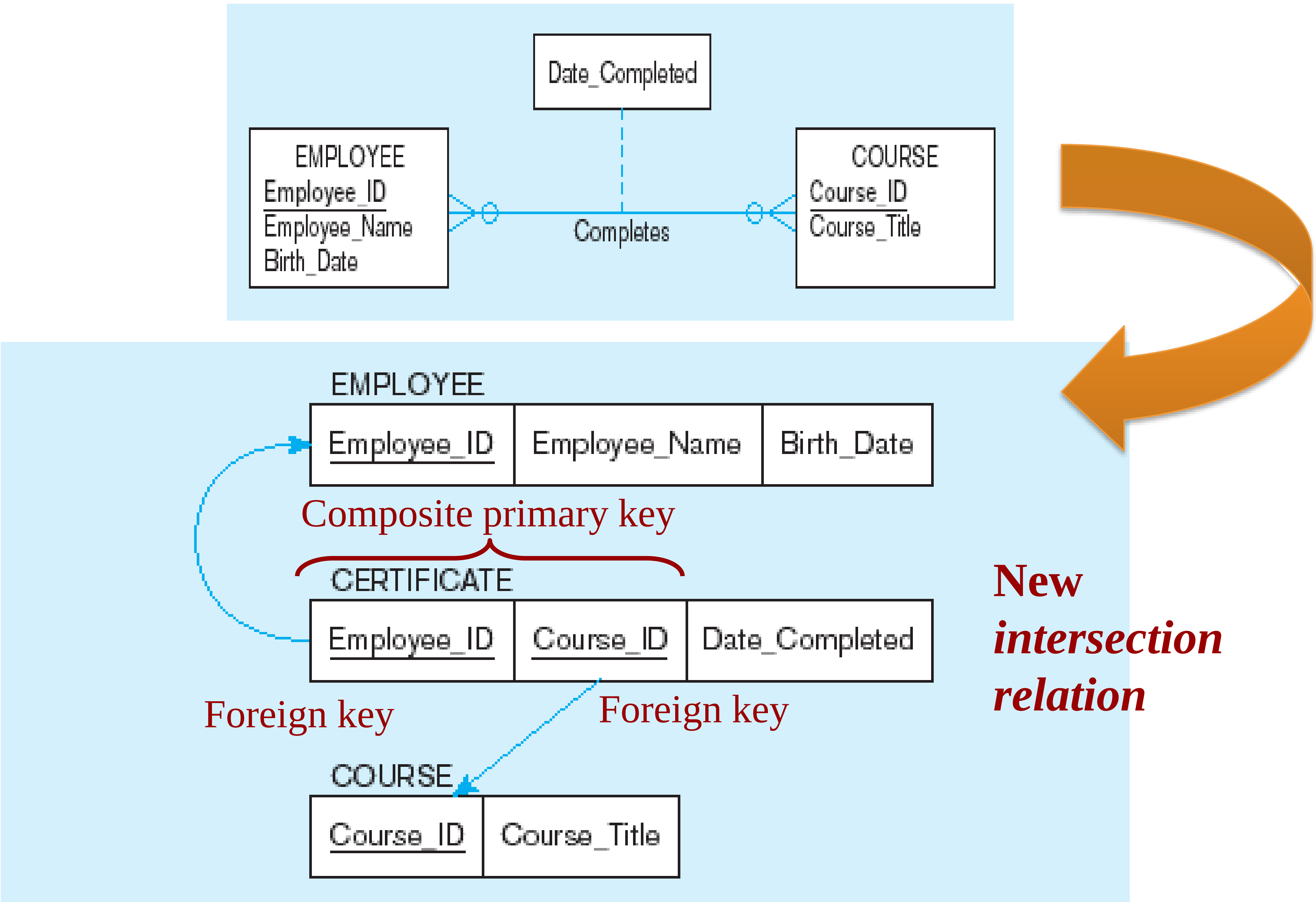



(b) Mapping the relationship

**[Primary key on the one side becomes a foreign key on the many side]**

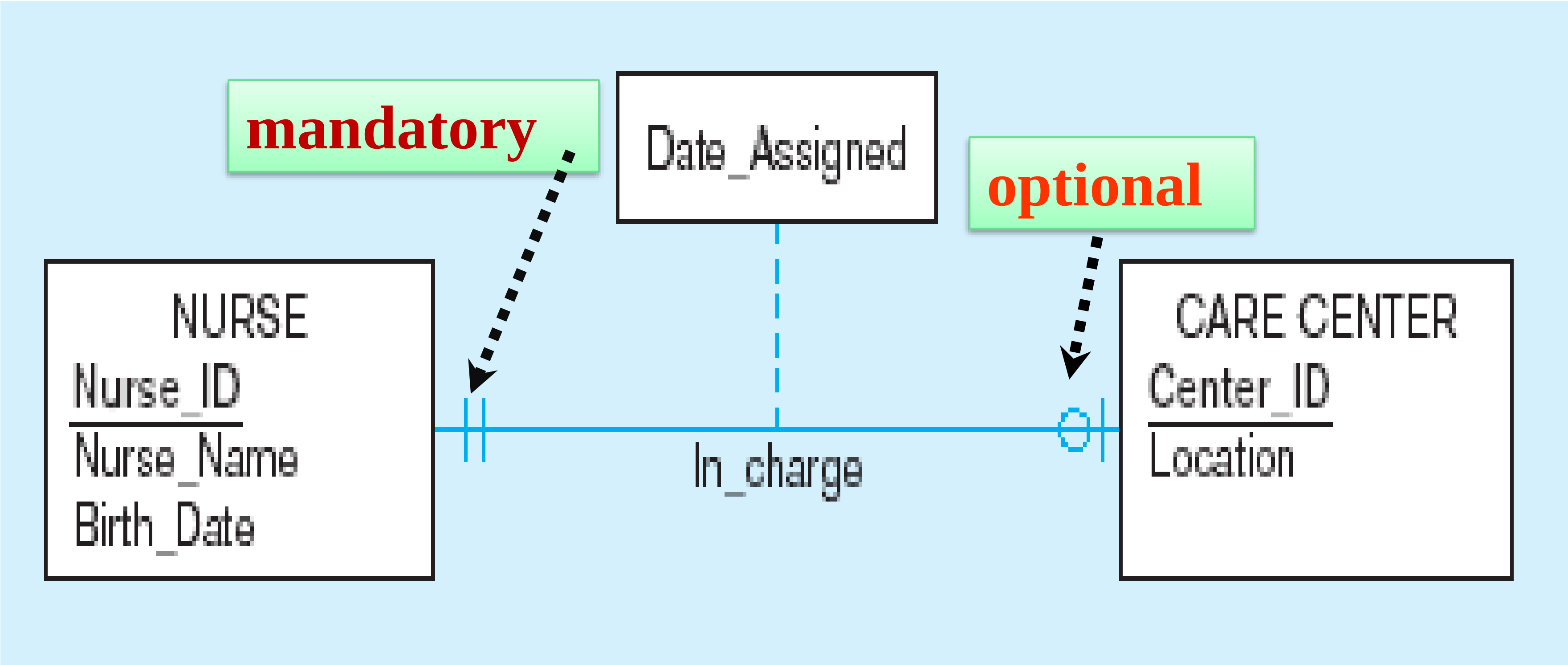


# Example of mapping an M:N relationship

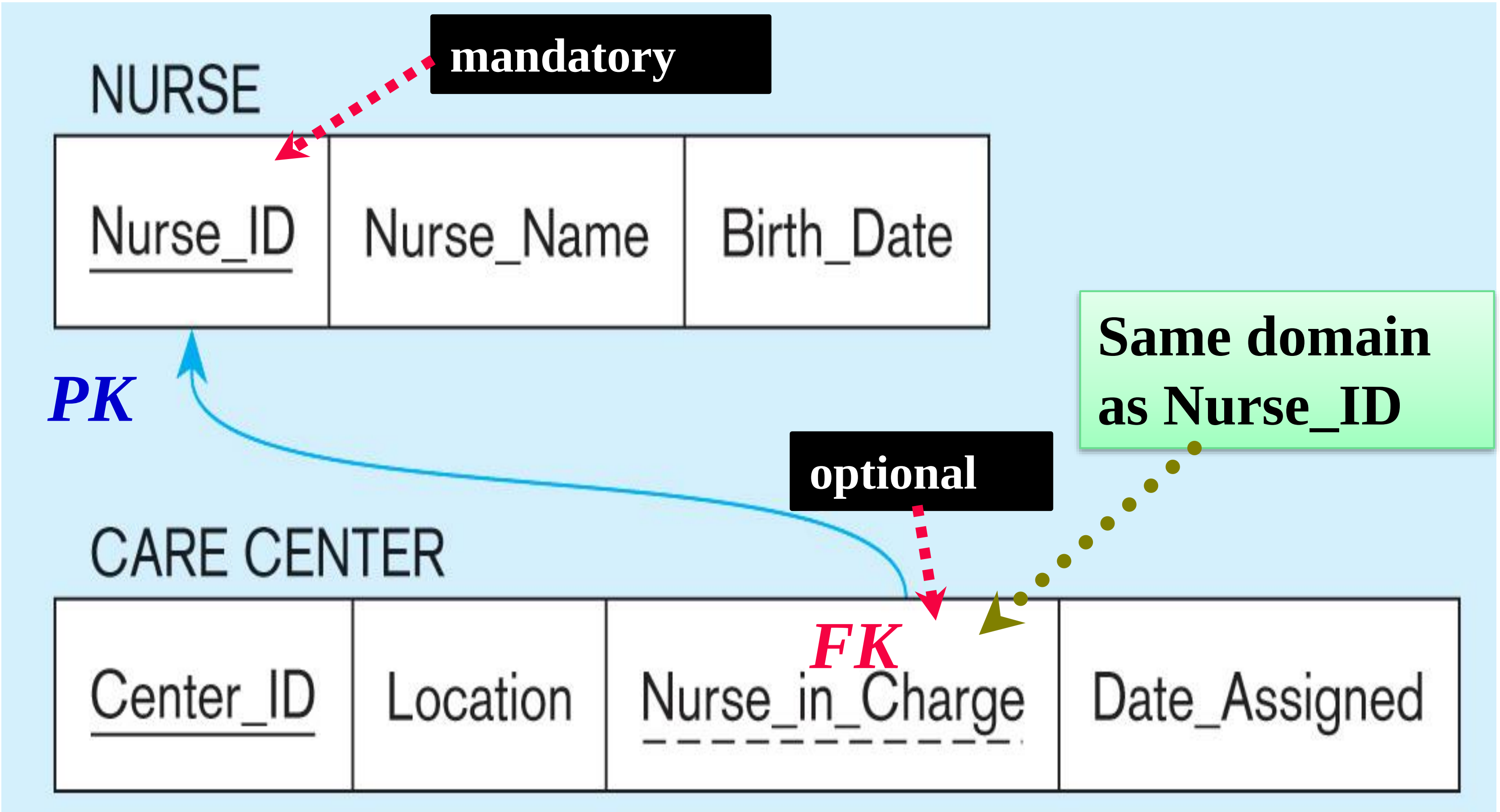


# Mapping a binary 1:1 relationship

*In_charge* relationship (1:1)



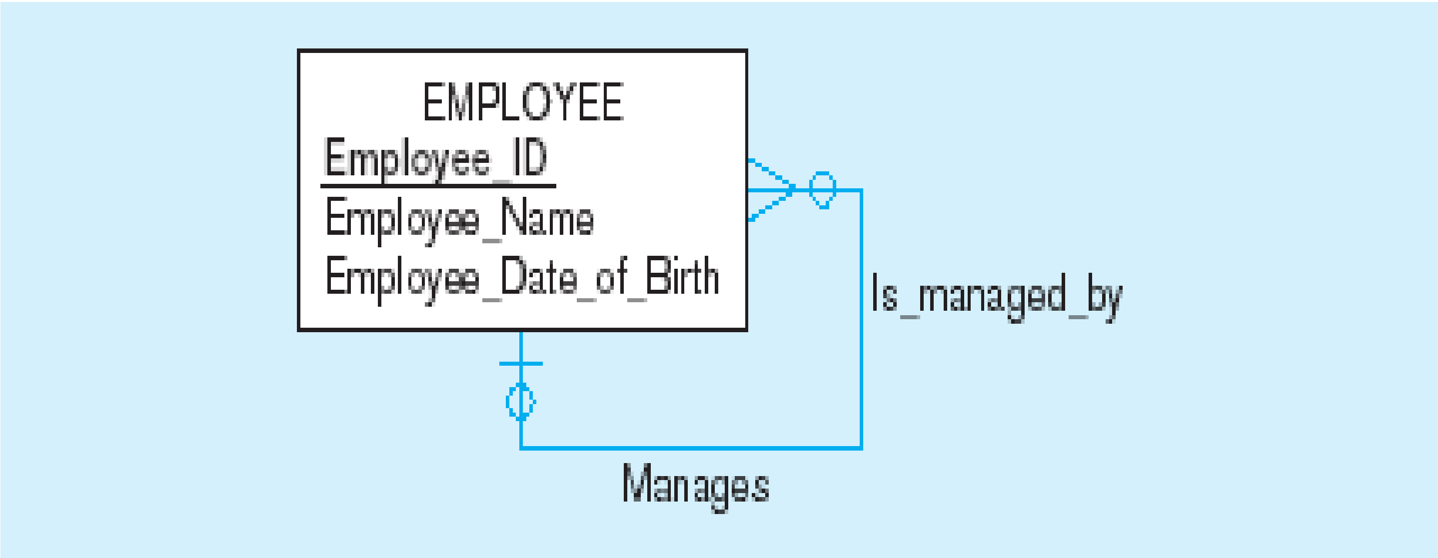
# Mapping 1:1 relationship - Resulting relations




[Primary key on the mandatory side becomes a foreign key on the optional side]

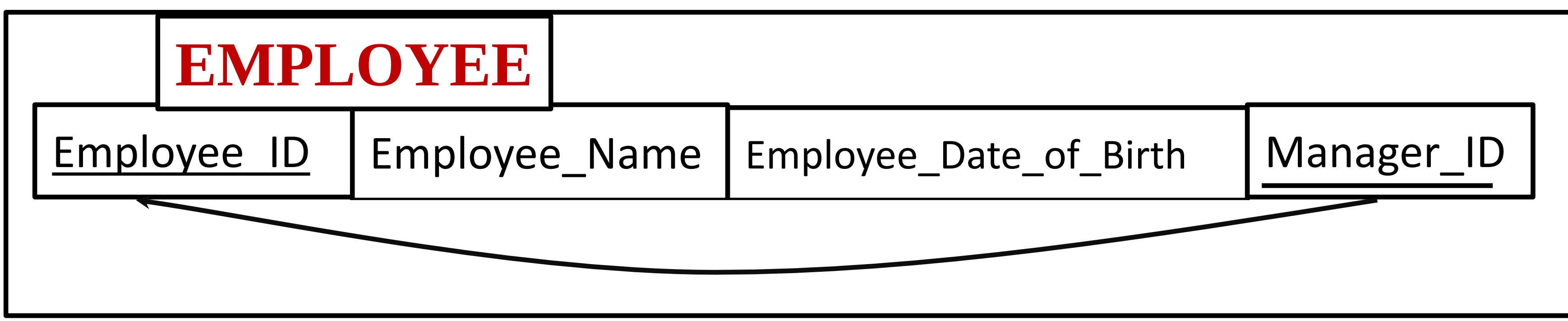
# Mapping a unary 1:N relationship

(a) EMPLOYEE entity with unary relationship



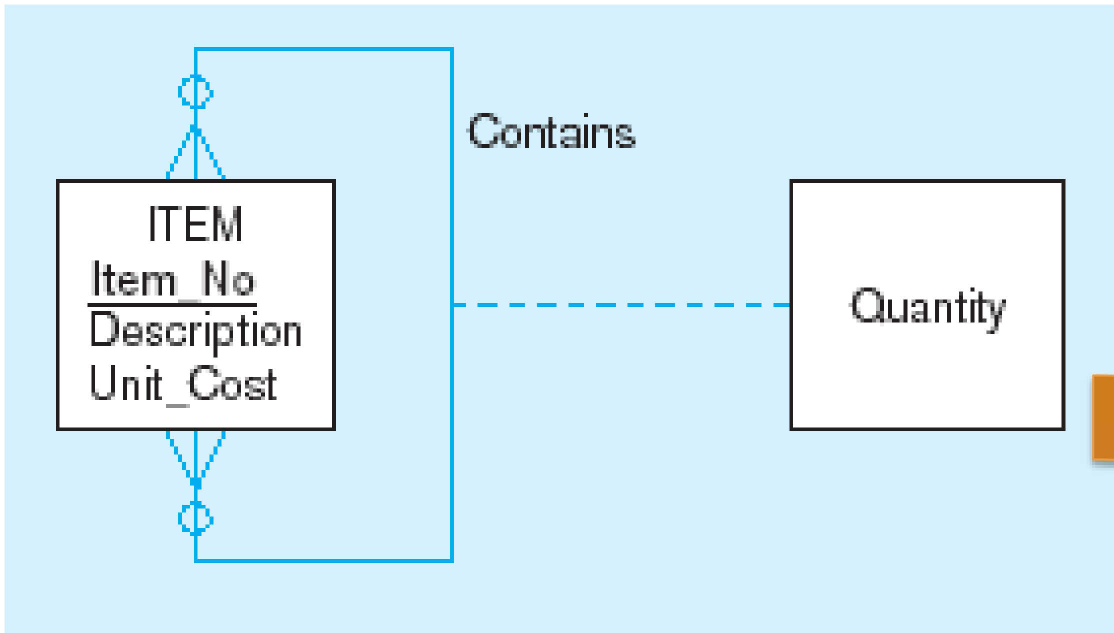

**A recursive FK is a FK in a relation that references the PK values of that same relation. It must have the same domain as the PK.**

(b) EMPLOYEE relation with recursive foreign key

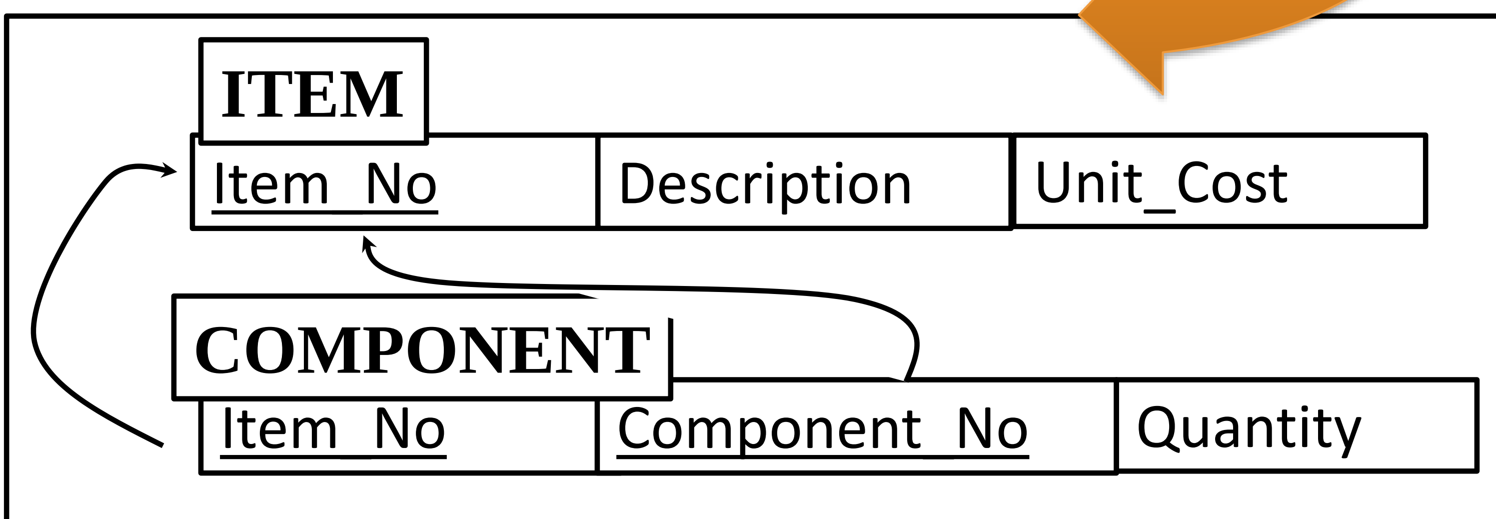


# Mapping a unary M:N relationship

(a) M:N unary relationship



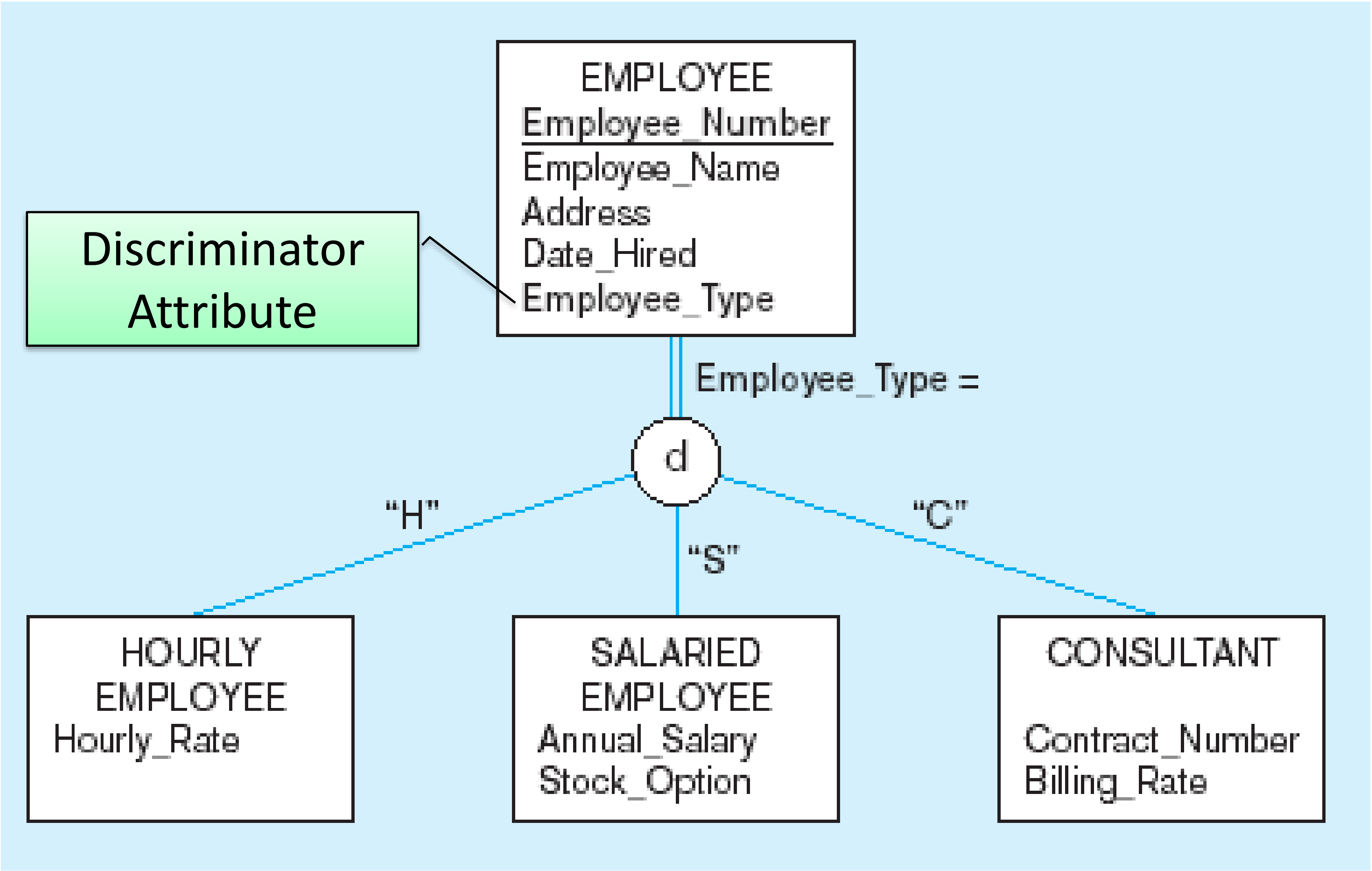

(b) ITEM and COMPONENT relations



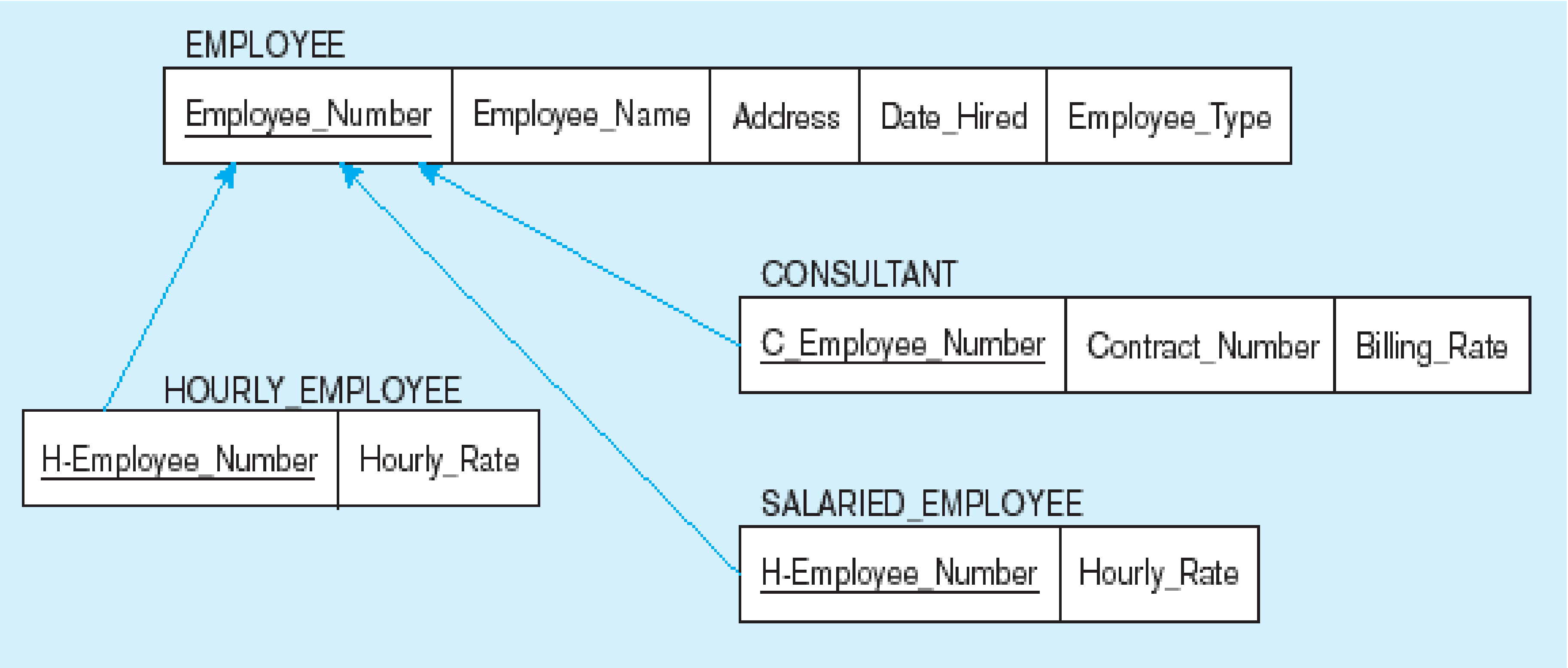
# Mapping Supertype/Subtype Relationships

- 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

# Supertype/subtype relationships



# Mapping Supertype/subtype relationships to relations




**These are implemented as one-to-one relationships**

# Normalization

# Data Normalization

- The process of decomposing relations with anomalies to produce smaller, well ***structured*** relations free of redundancy:
  - A relation that contains minimal data redundancy allows users to insert, delete, and update rows **without causing data inconsistencies**
- Normalization is a primarily tool to validate and improve a logical design so that it satisfies certain constraints that ***avoid unnecessary duplication of data***
- 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

| Student# | Advisor | Adv-Room |
|---|---|---|
| 1022 | Jones | 412 |
| 4123 | Smith | 216 |
| 4124 | Smith | 216 |

# Anomalies in this Table

- **Insertion** – can't enter a new student without having the Advisor Room
- **Deletion** – if we remove Student# 1022, we lose information about the room of the Advisor Jones
- **Modification** – Changing the room of the advisor Smith forces us to update multiple records

## Why do these anomalies exist?

Because two entity types were combined. This results in duplication, and an **unnecessary dependency between the entities**

# First Normal Form

- Every attribute value is atomic (singled-value)
- A table is in its first normal form if it contains **no repeating attributes** or groups of attributes

Students

| FirstName | LastName | Knowledge |
|---|---|---|
| Thomas | Mueller | Java, C++, PHP |
| Ursula | Meier | PHP, Java |
| Igor | Mueller | C++, Java |

Startsituation

Result after Normalisation

Students

| FirstName | LastName | Knowledge |
|---|---|---|
| Thomas | Mueller | C++ |
| Thomas | Mueller | PHP |
| Thomas | Mueller | Java |
| Ursula | Meier | Java |
| Ursula | Meier | PHP |
| Igor | Mueller | Java |
| Igor | Mueller | C++ |

To get to the first normal form (1NF) we must **create a separate tuple for each value of the multivalued attribute**

# Second Normal Form

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

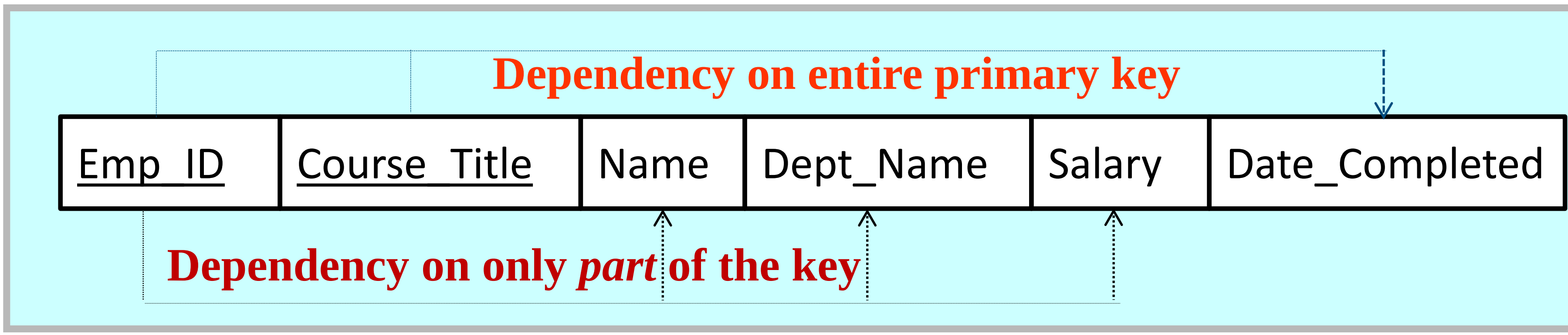
## EXAMPLE OF FUNCTIONAL DEPENDENCY

| <u>EMPLOYEE NUMBER</u> | <u>PROJECT CODE</u> | EMPLOYEE NAME | PROJECT NAME | TOTAL DAYS ON PROJECT |
|---|---|---|---|---|

| | |
|---|---|
| EMPLOYEE NAME | is functionally dependent on EMPLOYEE NUMBER but not on PROJECT CODE |
| PROJECT NAME | is functionally dependent on PROJECT CODE but not on EMPLOYEE NUMBER |
| TOTAL DAYS ON PROJECT | is functionally dependent on the concatenated key of EMPLOYEE NUMBER and PROJECT CODE |

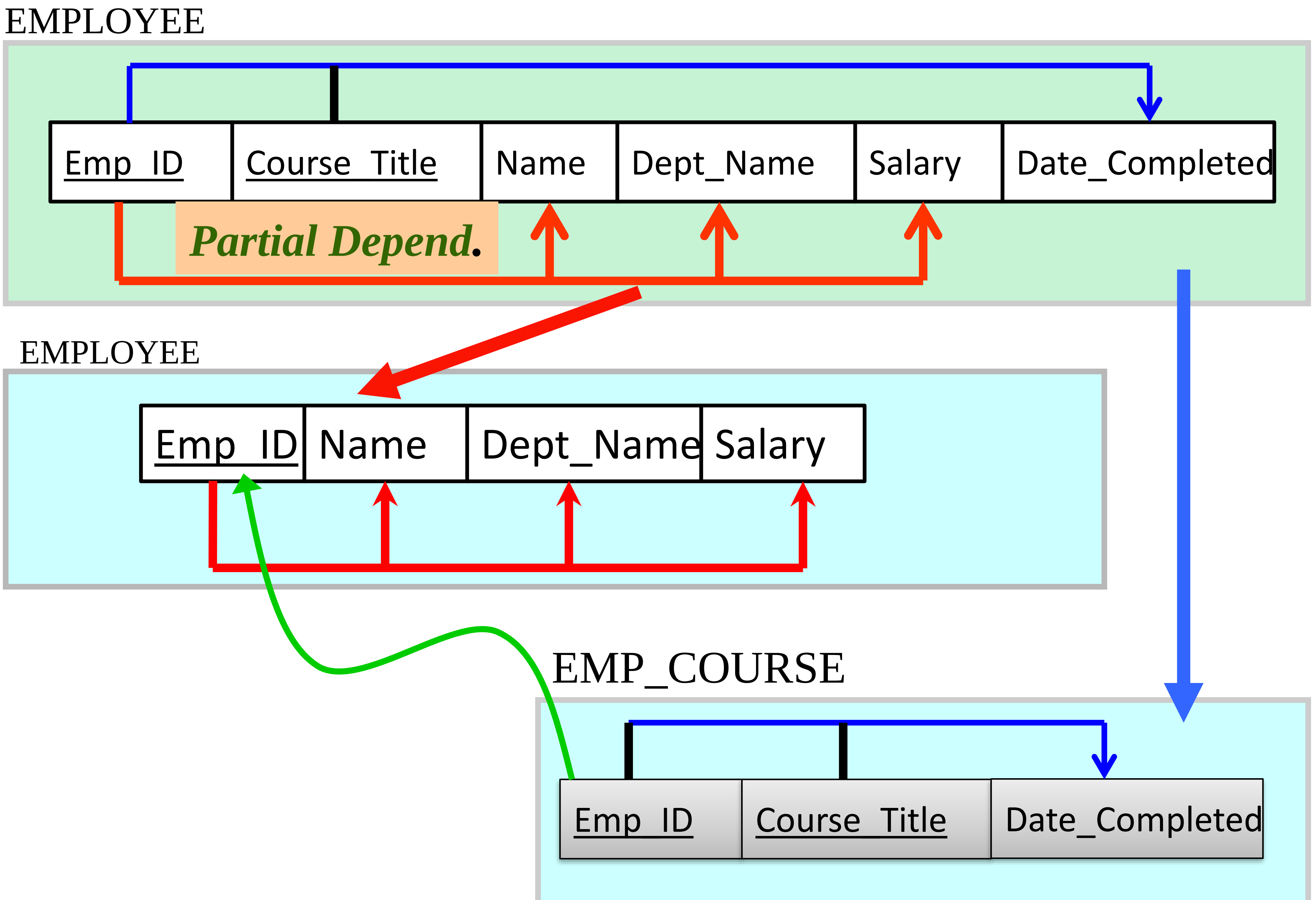
# Functional Dependencies in EMPLOYEE

Functional Dependencies in EMPLOYEE




**Therefore, NOT in 2$^{nd}$ Normal Form!!**

# Normalization from 1NF to 2NF

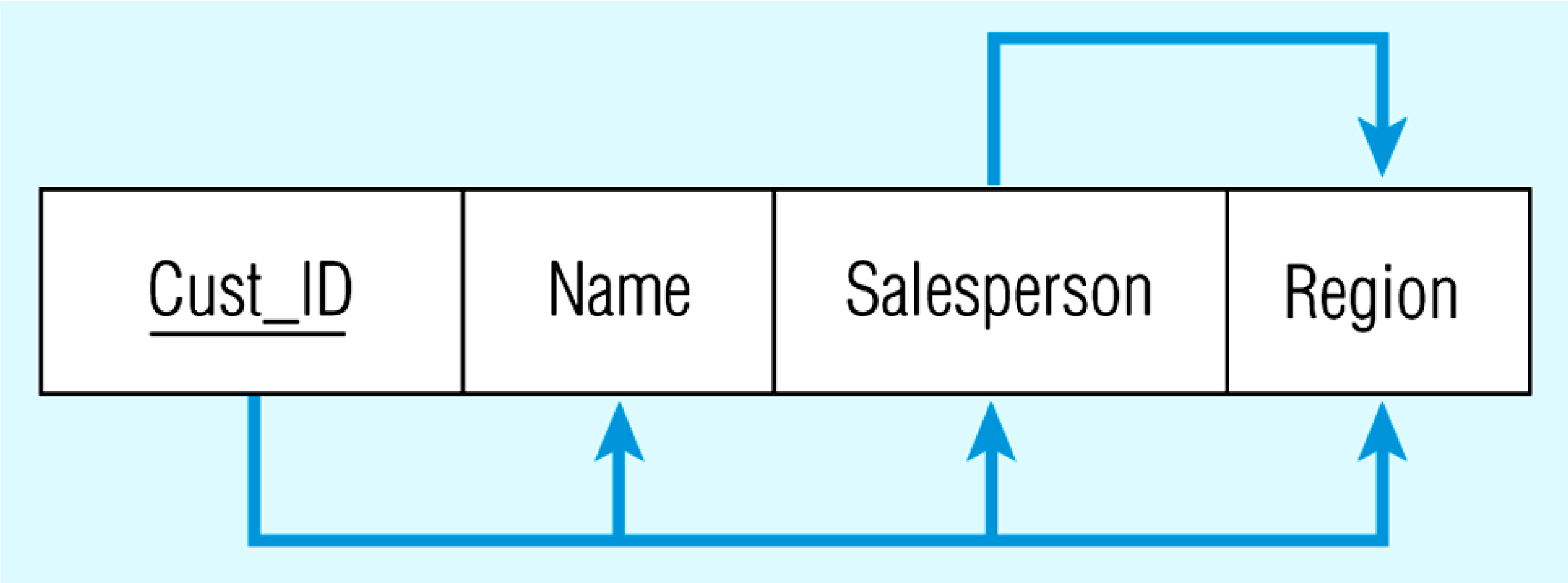


# Third Normal Form

- 2NF and no transitive dependencies (no functional dependency between non-key attributes)

=> all fields are functionally dependent ONLY on the primary key

# Relation with transitive dependency

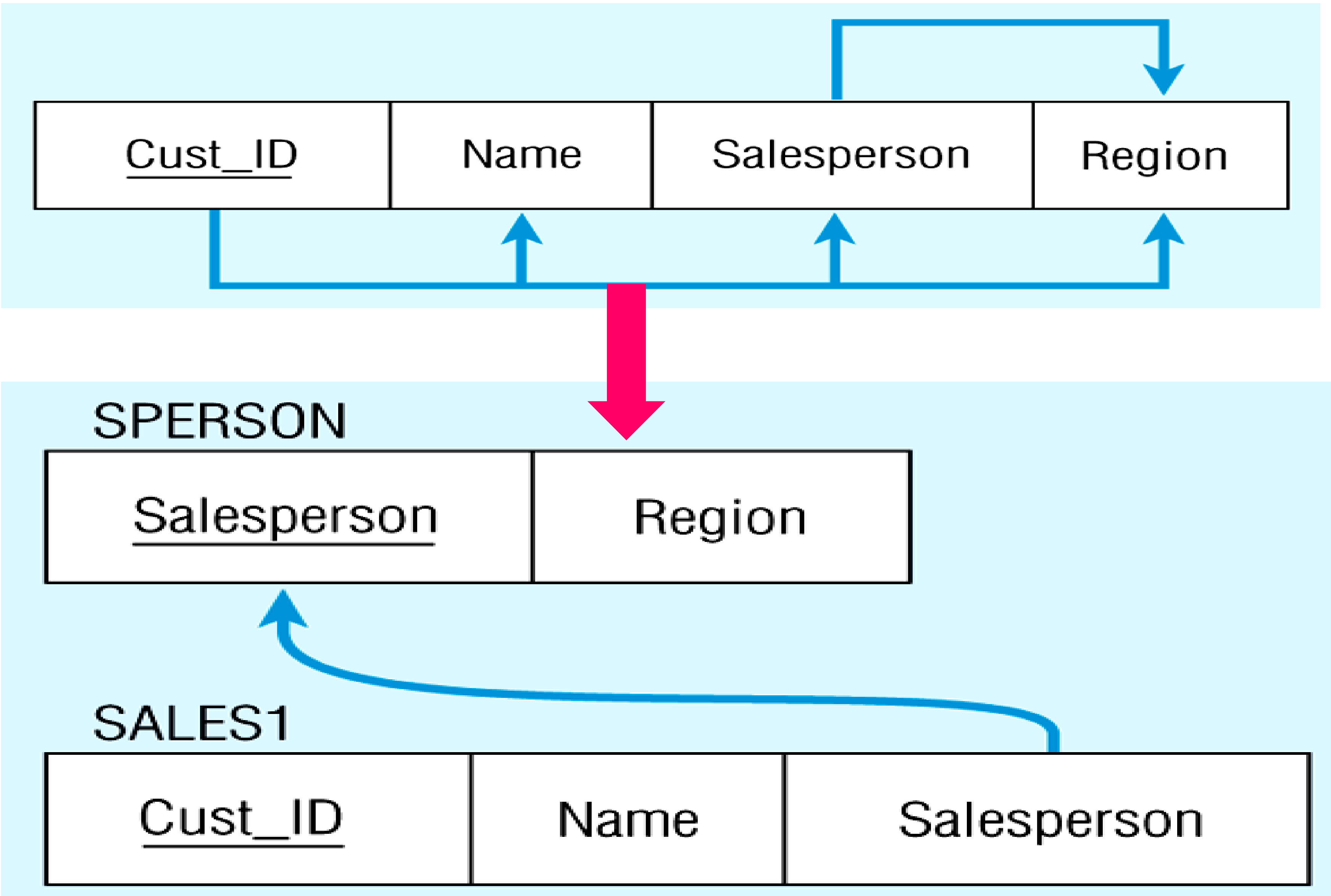



**CustID → Name**
**CustID → Salesperson**
**CustID → Region**
**and**
**Salesperson → Region**
**All this is OK**
**(2nd NF)**

**BUT**
**CustID → Salesperson → Region**
**implies**
**CustID → Region**
*Transitive dependency*
*(not in 3rd NF)*

# Relations in 3NF

**Remove a transitive dependency**



# Removing a transitive dependency

(a) Decomposing the SALES relation

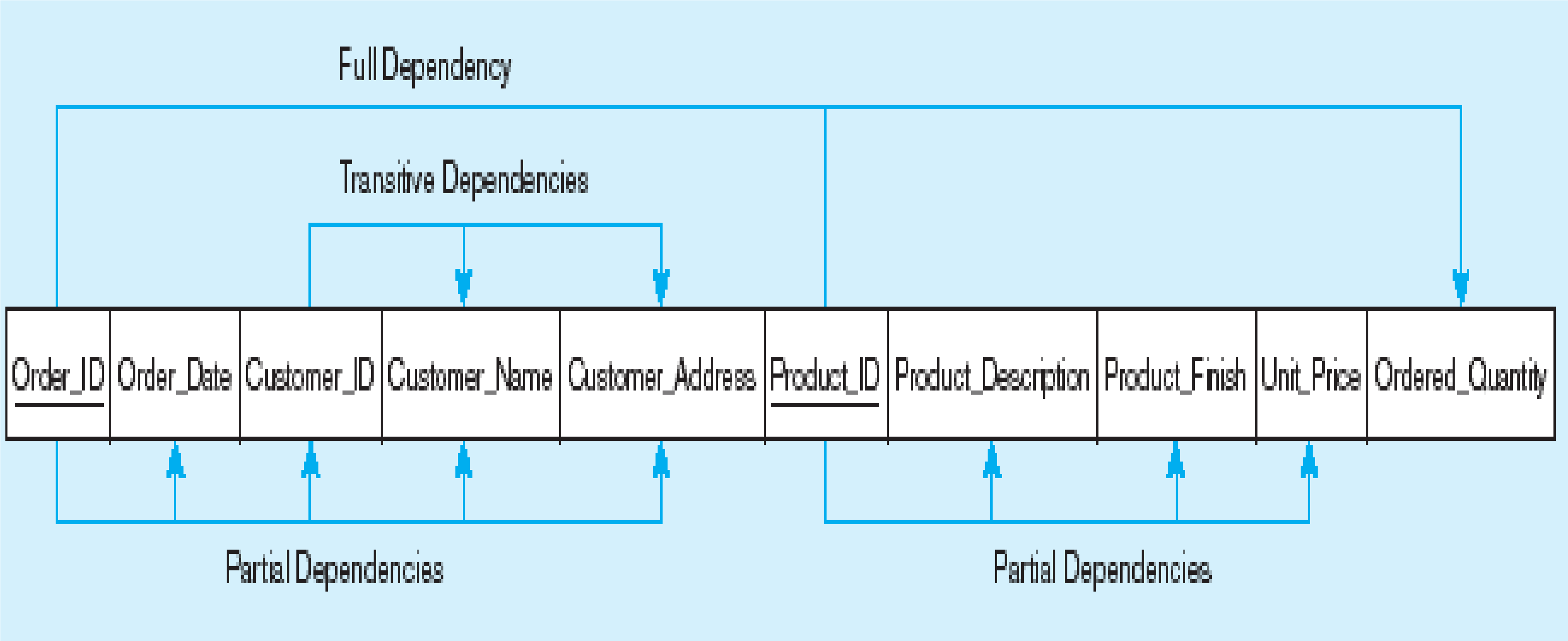
SALES

| Cust_ID | Name | Salesperson |
|---|---|---|
| 8023 | Anderson | 101 |
| 9167 | Bancroft | 102 |
| 7924 | Hobbs | 101 |
| 6837 | Tucker | 103 |
| 8596 | Eckersley | 102 |
| 7018 | Arnold | 104 |

S_PERSON

| Salesperson | Region |
|---|---|
| 101 | South |
| 102 | West |
| 103 | East |
| 104 | North |

# Functional dependency diagram for INVOICE




Order_ID ➔ Order_Date, Customer_ID, Customer_Name, Customer_Address

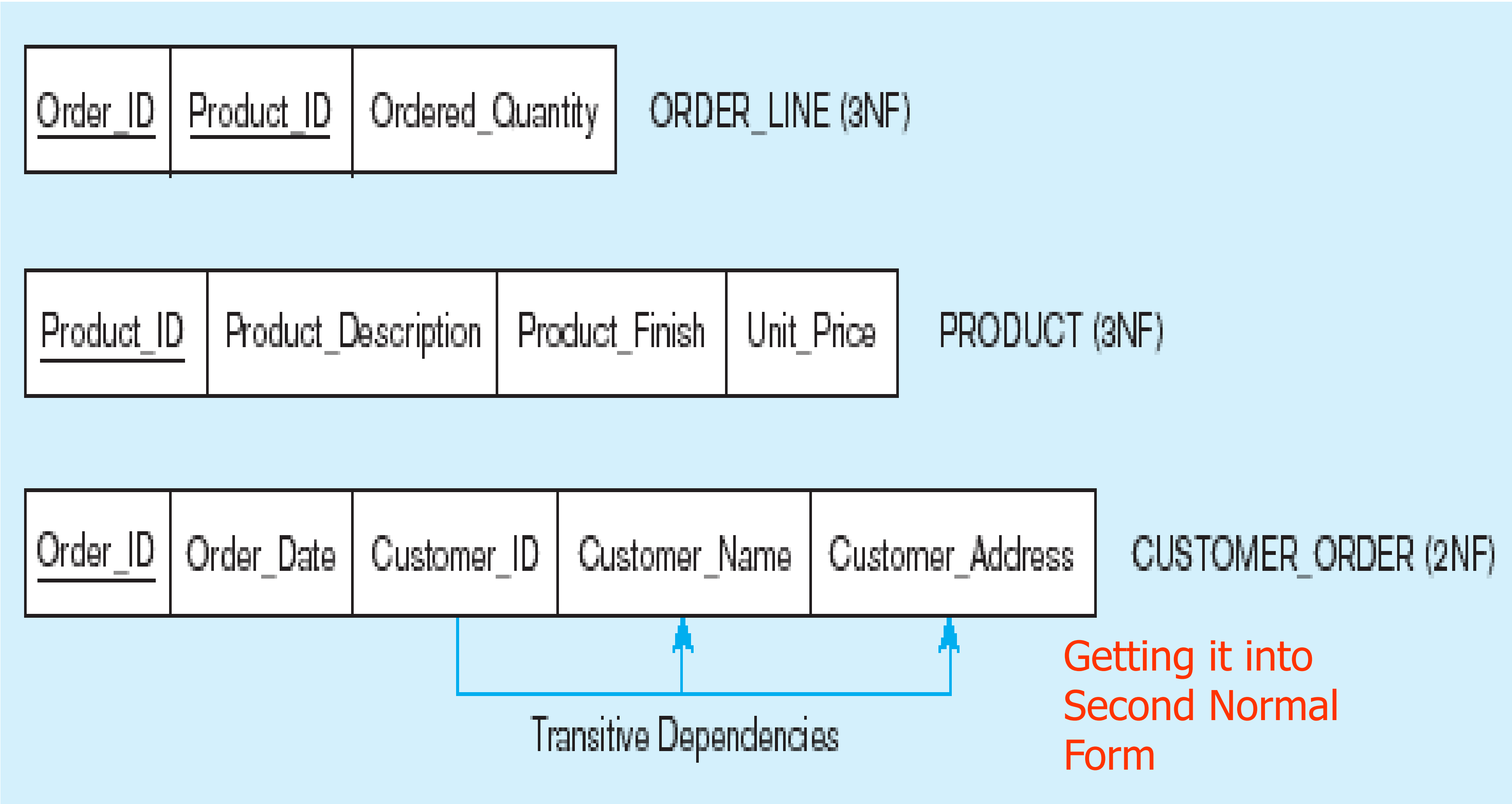
Customer_ID ➔ Customer_Name, Customer_Address

Product_ID ➔ Product_Description, Product_Finish, Unit_Price

Order_ID, Product_ID ➔ Order_Quantity

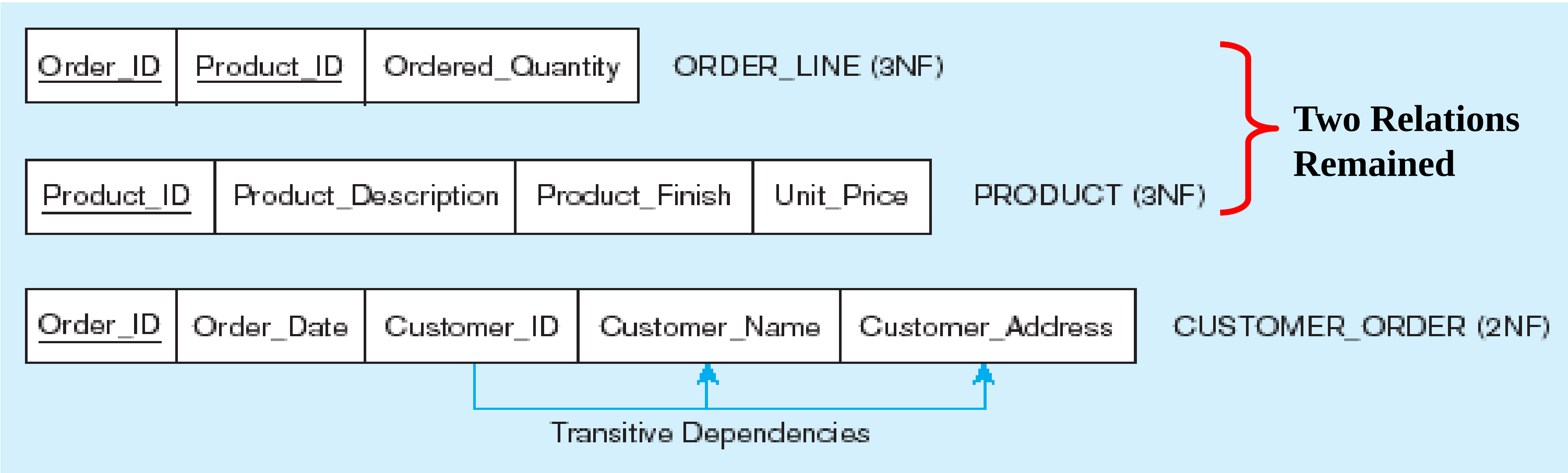
**Therefore, NOT in 2nd Normal Form**

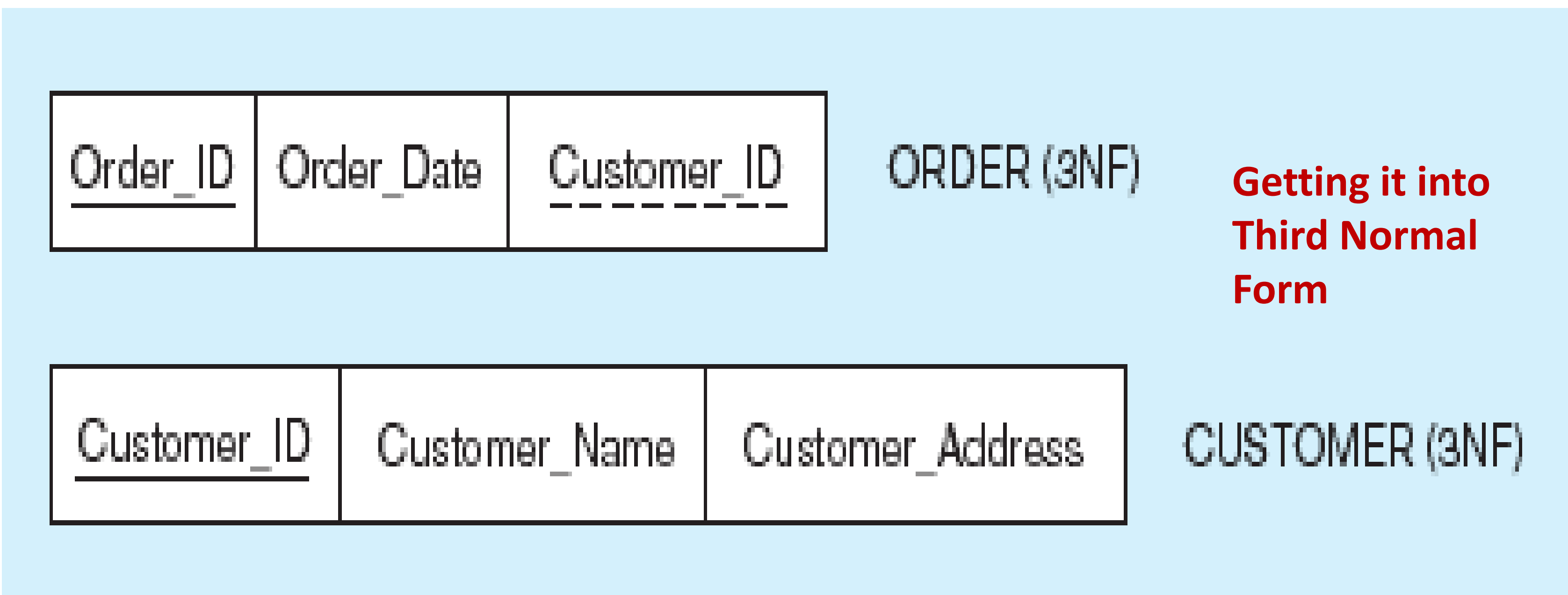
# Partial Dependencies were Removed (2NF)




Partial dependencies are removed, but there are still transitive dependencies

# Transitive Dependencies were Removed (3NF)



# 4th Normal Form

- No repeating group
- 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 ➔➔ B (B multi-valued depends on A)

# 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

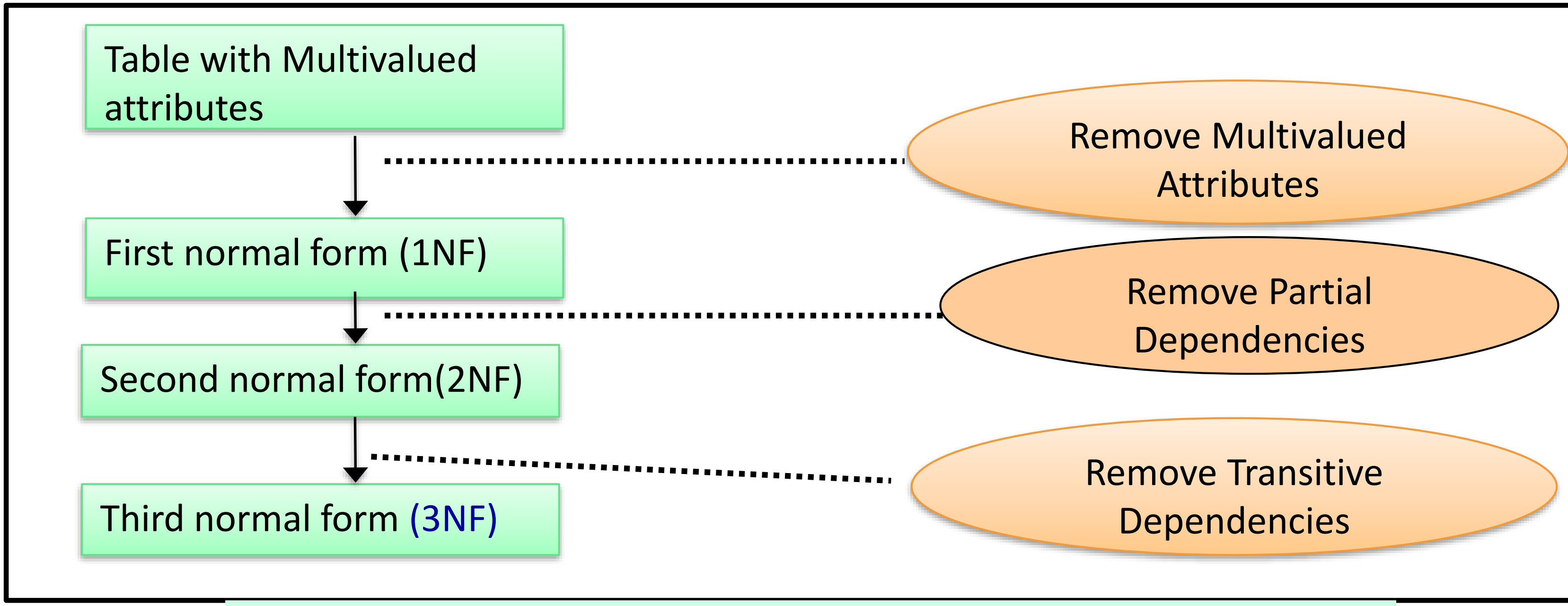
# Solution

- Solution: Decouple MVD to a separate table. Then, connect each to Student table

| Student |
|---|
| Sok |
| Sao |
| Chan |
| Puth |
| Tith |

| Student | Major |
|---|---|
| Sok | IT |
| Sao | IT |
| Sao | Med |
| Chan | IT |
| Puth | NULL |
| Tith | NULL |

| Student | Hobby |
|---|---|
| Sok | Football |
| Sok | Volleyball |
| Sao | Football |
| Chan | NULL |
| Puth | Football |
| Tith | NULL |




# The Normalisation Process

Normalisation

1. Check for multi-valued attributes
   If you find any, restructure table to remove them
   table is now in 1st NF
2. Check for partial dependencies
   If you find any, restructure table to remove them
   table is now in 2nd NF
3. Check for transitive dependencies
   If you find any, restructure table to remove them
   table is now in 3rd NF