# Conceptual Data Modeling using the (E)ER Model and UML Class Diagram

ACS 575: Database Systems

Instructor: Dr. Jin Soung Yoo
Department of Computer Science
Purdue University Fort Wayne

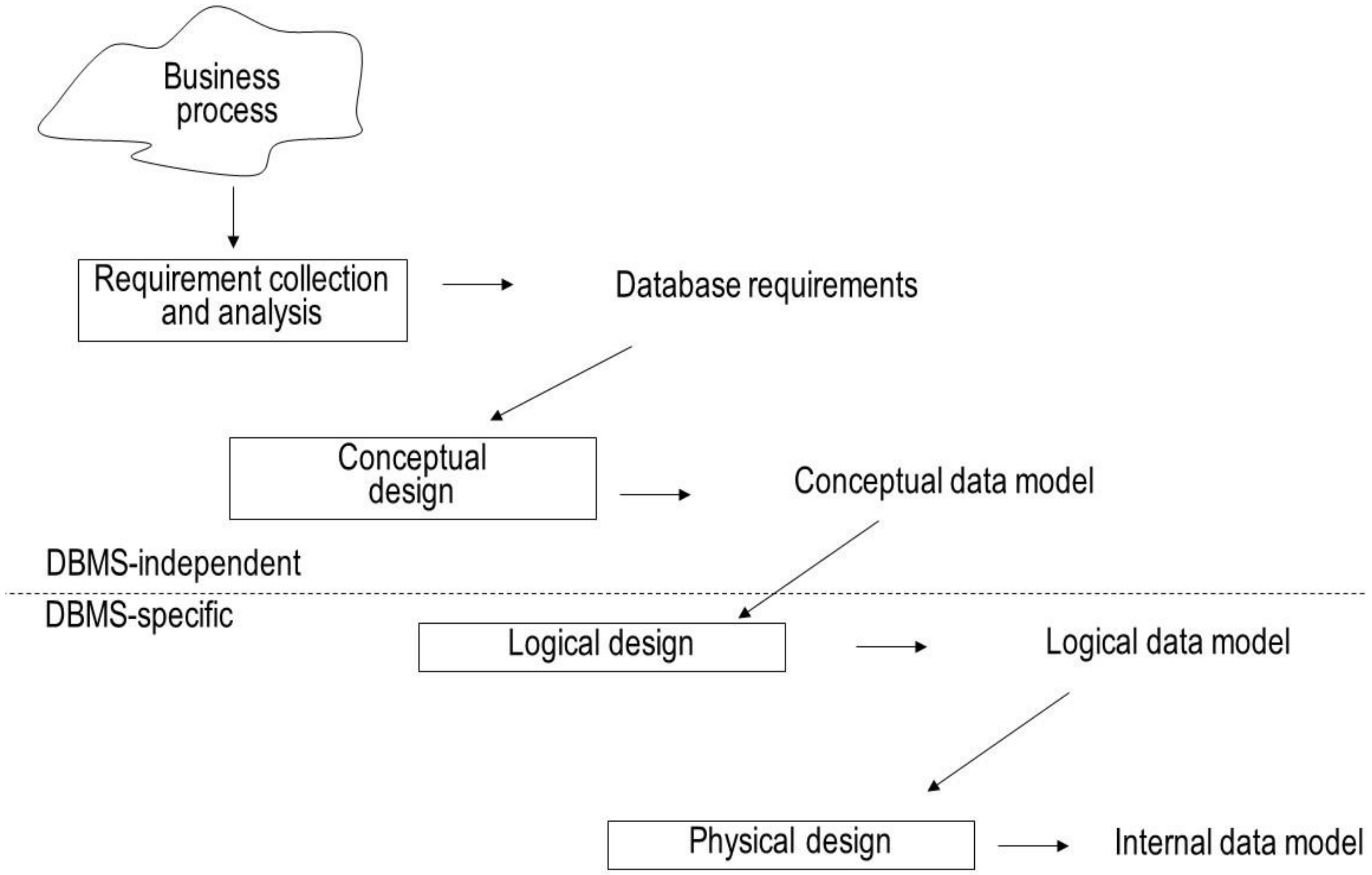
## References

- W. Lemanhieu, et al., Principles of Database Management, Ch 3

## Outline

- Phases of Database Design
- Entity Relationship (ER) Model
- Enhanced Entity Relationship (EER) Model
- UML Class Diagram
- Case Studies

## Phases of Database Design



## First Phase: Requirement Collection and Analysis

- The first step is **requirement collection and analysis**, where the aim is to carefully understand the different steps and data needs of the process.
- The information architect will collaborate with the business user to make the database requirements clear.
- Various techniques can be used, such as interviews or surveys with end-users, inspections of the documents used in the current process, etc.

## Second Phase: Conceptual Design

- During the **conceptual design**, information architects and business users try to formalize the data requirements in a conceptual data model.
- The **conceptual model** should be a high-level model, meaning it should be both easy to understand for the business user and formal enough for the database designer who will use it in the next step.
- The conceptual data model must be user-friendly, and preferably have a graphical representation such that it can be used as a handy communication and discussion instrument between both information architects and business users.

## Second Phase: Conceptual Design (cont.)

- The conceptual data model should be flexible enough that new or changing data requirements can easily be added to the model.
- It must be DBMS- or implementation-independent since its only goal is to adequately and accurately collect and analyze data requirements.
- This conceptual model will also have its limitations, which should be clearly documented and followed up during application development.

## Third Phase: Logical Design

- Once all parties have agreed upon the conceptual data model, it can be mapped to a **logical data model** by the database designer during the **logical design** step.
- The logical data model is based upon the data model used by the implementation environment.
- Although at this stage it is already known what type of DBMS (e.g., RDBMS, OODBMS, etc.) will be used, the product itself (e.g., Microsoft, IBM, Oracle) has not been decided yet.
- The mapping from a conceptual model to its logical model can result in a loss of semantics which should be properly documented and followed up during application development.

## Third Phase: Logical Design (Cont.)

- It might be possible that additional semantics can be added to further enrich the logical data model.
- Also, the *views* of the external data model can be designed during this logical design step.

## Final Phase: Physical Design

- In a **physical design step**, the logical data model can be mapped to an **internal data model** by the database designer.
- The DBA can also give some recommendations regarding performance during this physical design step.
- In this step, the DBMS product is known, the database objects based on the logical model are created, and the definitions are stored in the catalog.
- The database can then be populated with data and is ready for use.
- Again, any semantics lost or added during this step should be documented and followed

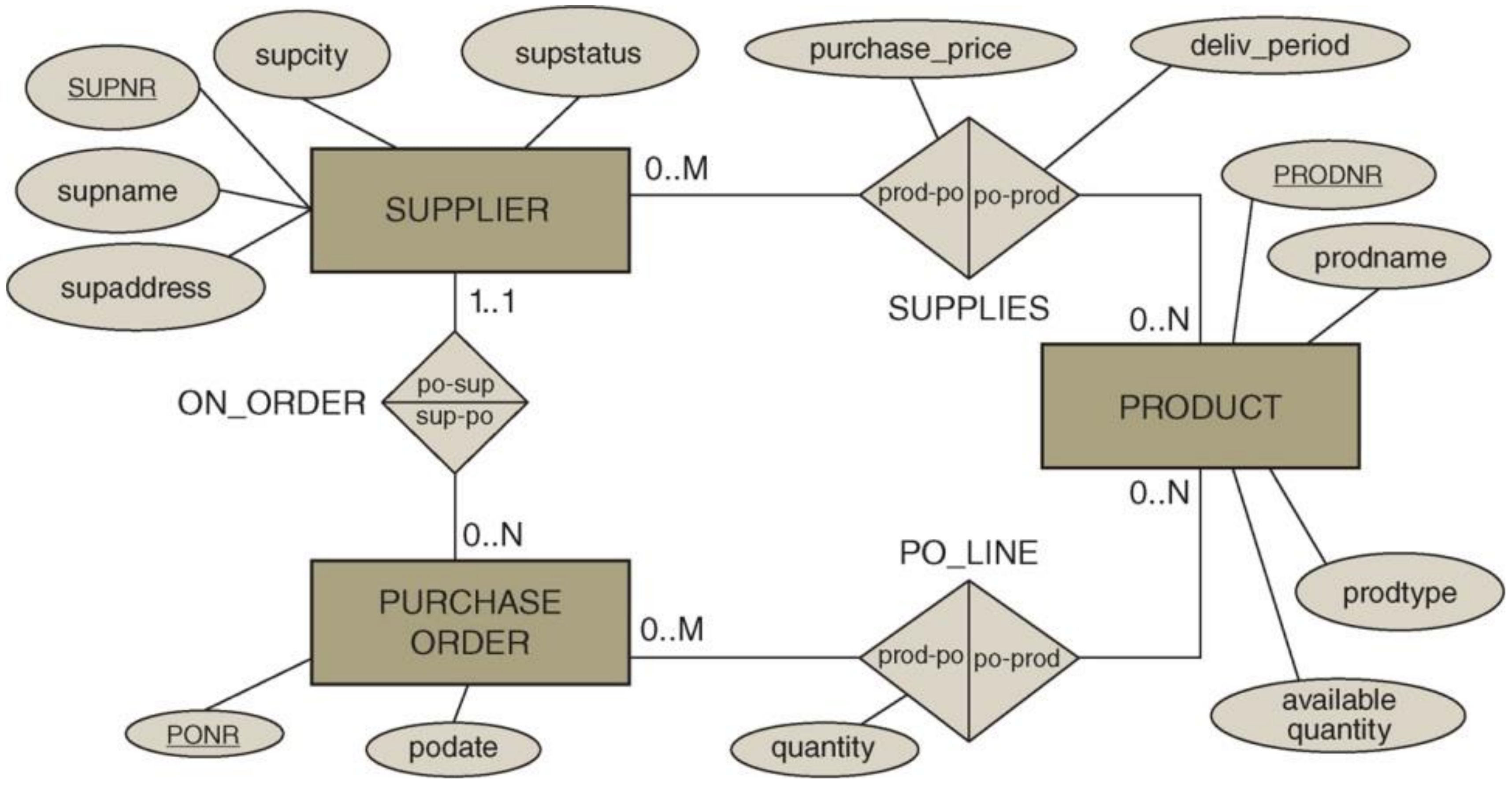
## Outline

- Phases of Database Design
- **Entity Relationship (ER) Model**
  - Entity Types
  - Attribute Types
  - Relationship Types
  - Weak Entity Types
  - Ternary Relationship Types
  - Limitations of the ER Model
- Enhanced Entity Relationship (EER) Bodel
- UML Class Diagram
- Case Studies

## Entity Relationship Model

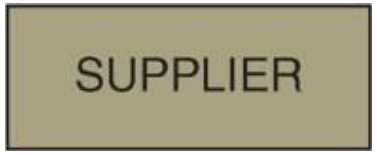
- The **Entity Relationship (ER) model** was introduced and formalized by Peter Chen in 1976.
- The ER model is one of the most popular data models for conceptual data model.
- Three building blocks of ER model are:
  - Entity types
  - Attribute types
  - Relationship types

# An Example of ER Diagram



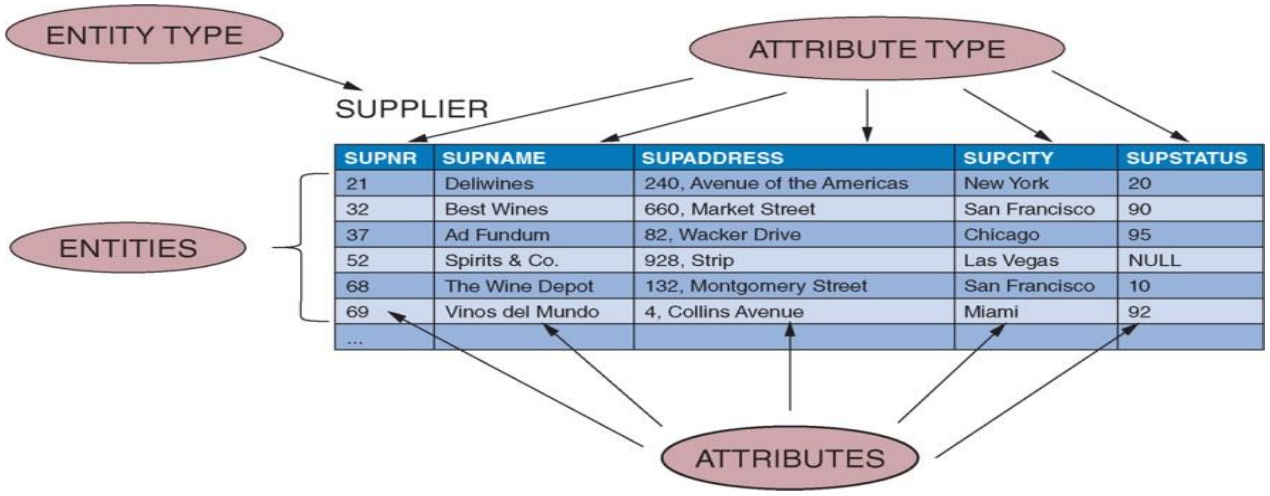
## Entity Types

- **Entity type** represents a business concept with an unambiguous meaning to a particular set of users
  - E.g., supplier, student, product or employee
- **Entity** is one particular occurrence or instance of an entity type
  - E.g., Best Wines and Ad Fundum are entities from the entity type supplier
- An entity type defines a collection of entities that have similar characteristics
- Entity types, not individual entities, are presented in ERD using a *rectangle*.



## Attribute Types

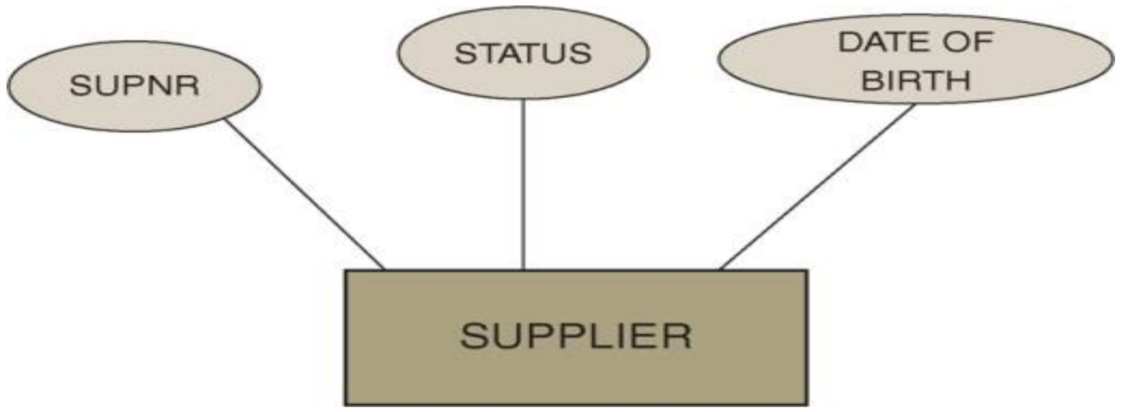
- **Attribute type** represents a property of an entity type.
  - E.g., name and address are attribute types of the entity type supplier
- **Attribute** is an instance of an attribute type
  - A particular entity has a value for each of its attribute types
- <u>An attribute type defines a collection of similar attributes</u>



| SUPNR | SUPNAME | SUPADDRESS | SUPCITY | SUPSTATUS |
|---|---|---|---|---|
| 21 | Deliwines | 240, Avenue of the Americas | New York | 20 |
| 32 | Best Wines | 660, Market Street | San Francisco | 90 |
| 37 | Ad Fundum | 82, Wacker Drive | Chicago | 95 |
| 52 | Spirits & Co. | 928, Strip | Las Vegas | NULL |
| 68 | The Wine Depot | 132, Montgomery Street | San Francisco | 10 |
| 69 | Vinos del Mundo | 4, Collins Avenue | Miami | 92 |
| ... | | | | |

## Attribute Types (Cont.)

- In the ER model, we focus on attribute types and not on individual attributes.
- Attribute types are represented using *ellipses*.



## Domains of Attributes

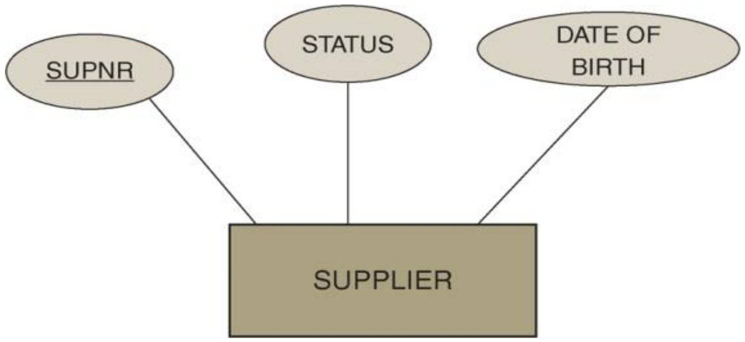
- The **domain** of an attribute specifies the set of values that may be assigned to the attribute for each individual entity
  - E.g., A domain of gender has two values: male or female.
  - A domain can have a format, e.g., dd/mm/yyyy for date
- A domain can also contain null values
  - A **null** value means the value is not known, not applicable or not relevant
- By convention, domains are not displayed in an ER model

## Different Attribute Types

- Key Attribute Types
- Simple vs. Composite Attribute Types
- Single-Valued vs. Multi-Valued Attribute Types
- Derived Attribute Type

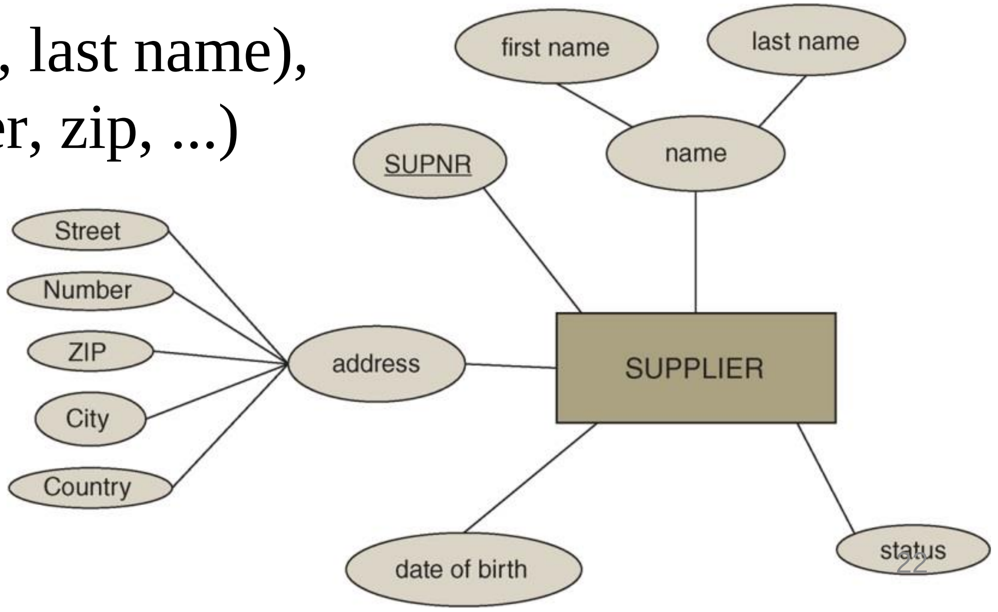
## Key Attribute Types

- A **key attribute type** is an attribute type whose values are distinct for each individual entity
  - E.g., supplier number, product number, social security number
  - A key attribute types can be used to uniquely identify each entity
- A key attribute type can also be a combination of attribute types
  - E.g., Depending on the business setting, a combination of flight number and departure date
- A key attribute type in ERD is presented using a *underline.*



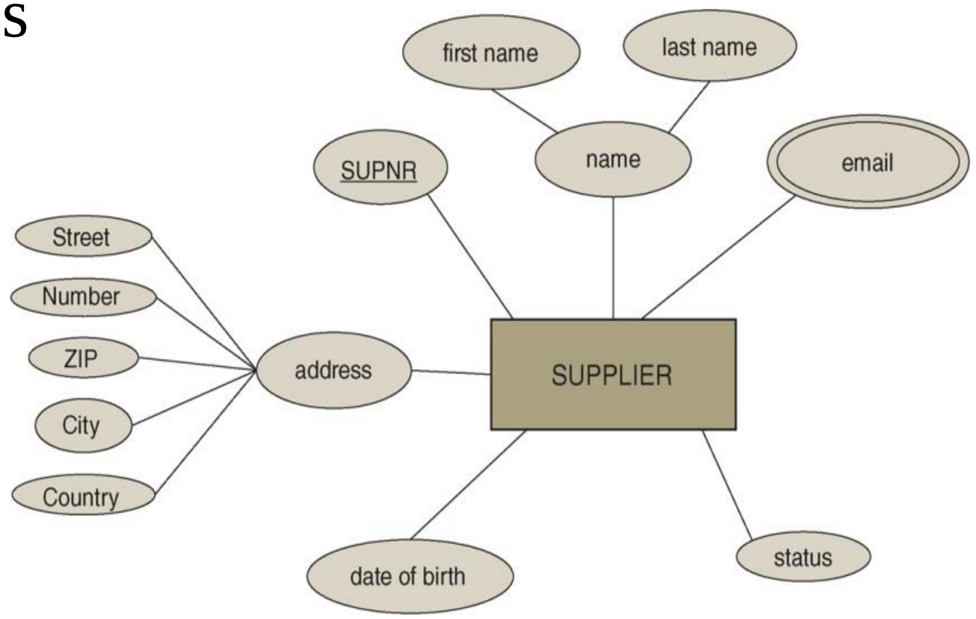
## Simple vs. Composite Attribute Types

- A **simple** or **atomic attribute type** cannot be further divided into parts
  - E.g., supplier number, supplier status
- A **composite attribute type** is an attribute type that can be decomposed into other meaningful attribute types
  - E.g., name (first name, last name), address (street, number, zip, ...)
  - In ERD, the overals of the component attribute types of an attribute type are connected to the attribute type's overal.



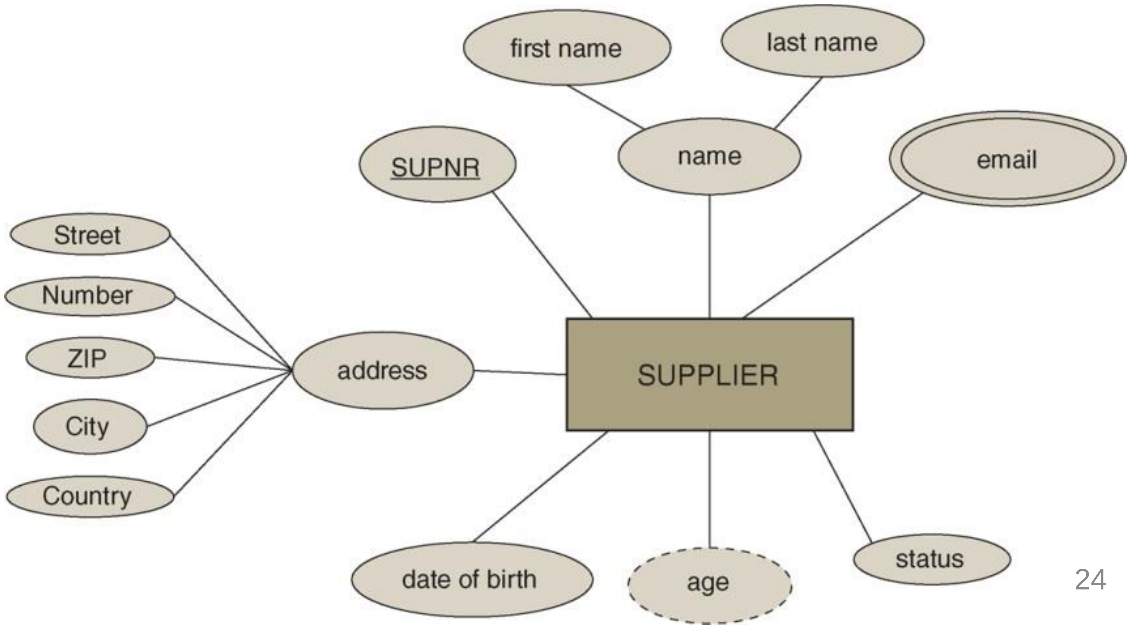
## Single-Valued vs. Multi-Valued Attribute Types

- A **single-valued attribute type** has only one value for a particular entity
  - E.g., product number, product name
- A **multi-valued attribute type** is an attribute type that can have multiple values
  - E.g., email address
- Multi-valued attribute types are represented using a *double ellipse* in the ER model



## Derived Attribute Types

- A **derived attribute type** is an attribute type which can be derived from another attribute type
  - E.g., age which can be derived from birth date
- Derived attribute types are represented using a *dotted ellipse* in the ER model

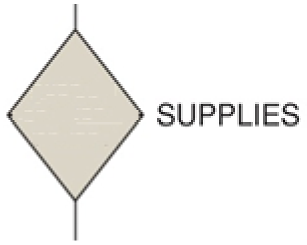


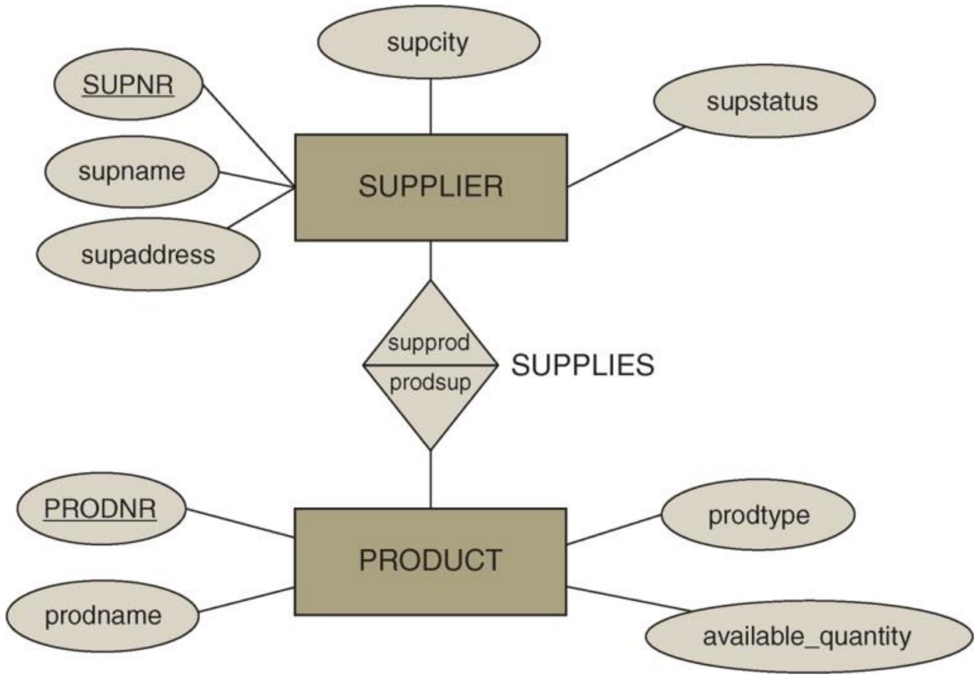
## Outline

- Phases of Database Design
- Entity Relationship (ER) Model
  - Entity Types
  - Attribute Types
  - ☞ **Relationship Types**
  - Weak Entity Types
  - Ternary Relationship Types
  - Limitations of the ER Model
- Enhanced Entity Relationship (EER) Model
- UML Class Diagram
- Case Studies

## Relationship Types

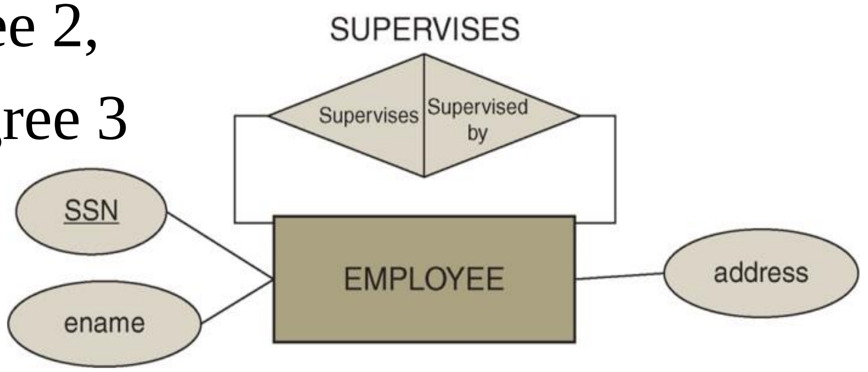
- A **relationship** represents an association between two or more entities
- A **relationship type** then defines a set of relationships among instances of one, two or more entity types
- Relationship types are indicated using a *rhombus* symbol in the ER model

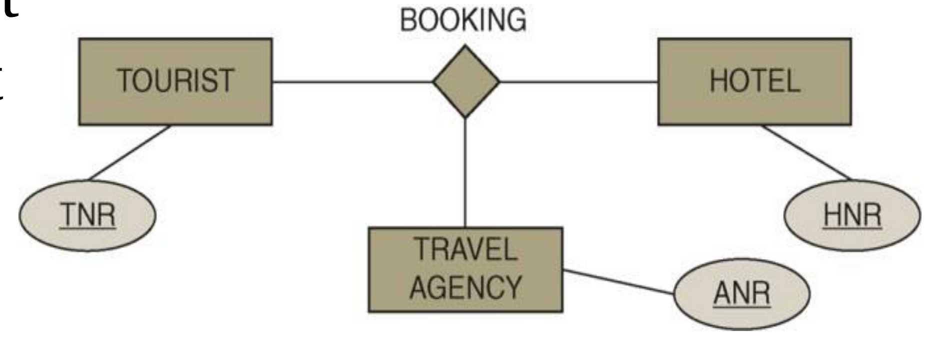




## Degree and Roles of Relationship Type

- The **degree** of a relationship type corresponds to the number of entity types participating in the relationship type
  - Unary relationship - degree 1,
  - Binary relationship - degree 2,
  - Ternary relationship - degree 3
- The **roles** of a relationship type indicate the various directions that can be used to interpret it
  - e.g., supervises/ supervised by

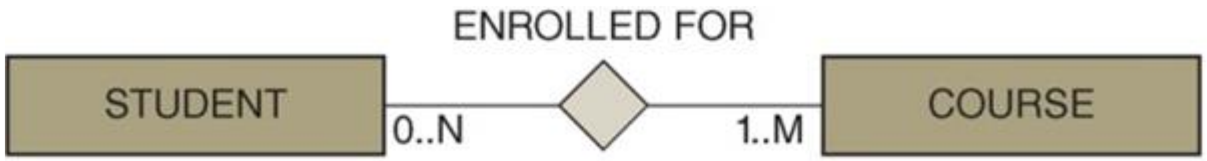


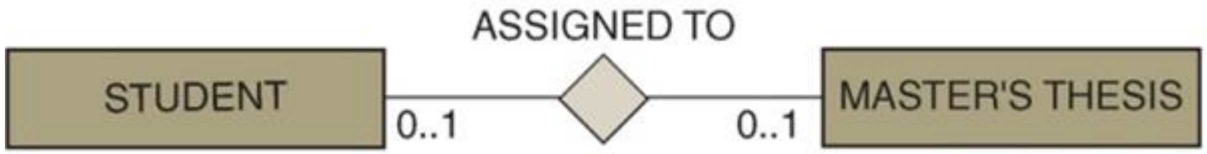


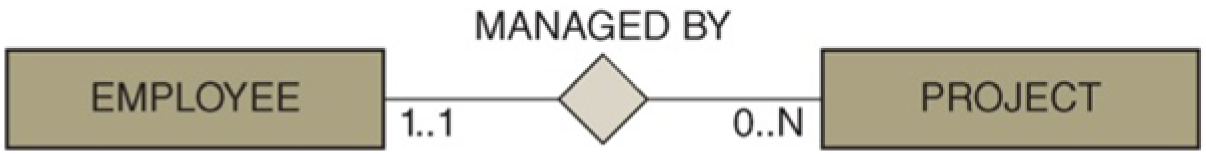
## Cardinalities of Relationship Type

- Every relationship type can be characterized in terms of its **cardinalities**, which specify the minimum or maximum number of relationship instances that an individual entity can participate in.
- **Minimum cardinality** can either be **0** or **1**
  - If **0**, **partial participation –** some entities may not participate in the relationship.
  - If **1**, **total participation** or **existence dependency**
- **Maximum cardinality** can either be **1** or **N**
  - If **1**, an entity can be connected to **at most one other entity** through that relationship type.
  - If **N** (an arbitrary integer number bigger than 1), an entity can be connected to **at most N other entities** by means of the relationship type.

## Examples of Relationship Cardinality

- 

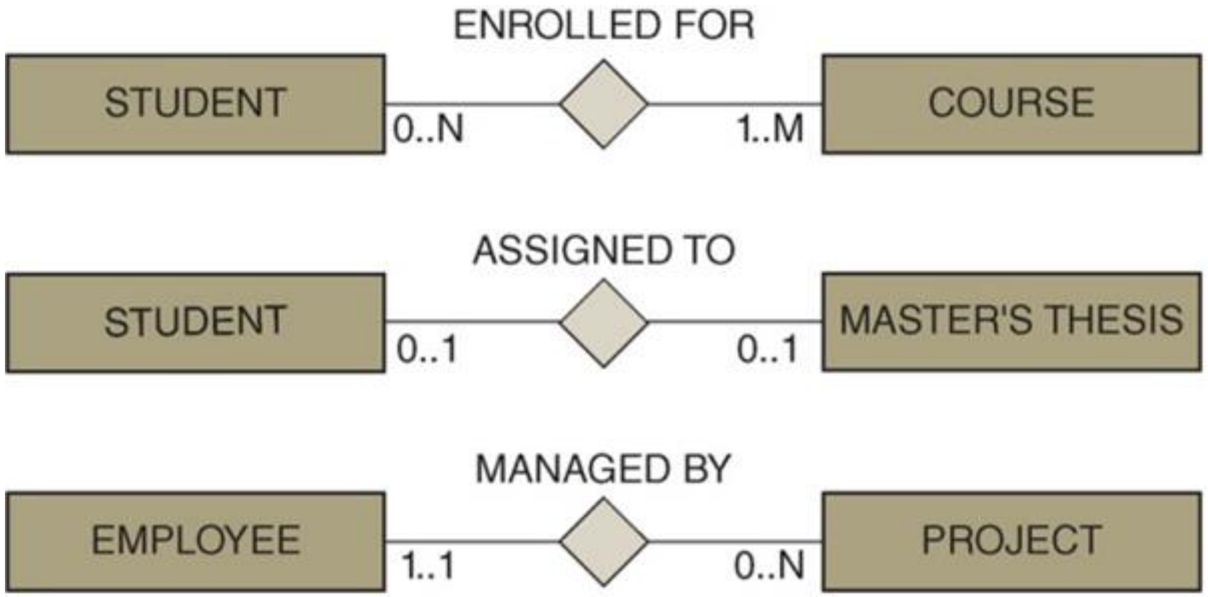
  - A student can be enrolled for a minimum of one course and a maximum of M courses. Conversely, a course can have minimum zero and maximum N students enrolled.
- 

  - A student can be assigned to minimum zero and maximum one master’s thesis. A master’s thesis is assigned to minimum zero and maximum one student.
- 

  - An employee can manage minimum zero and maximum N projects. A project is managed by exactly one employee (i.e., minimum one and maximum one)

## Relationship Types By Maximum Cardinality

- Relationship types are often characterized by their maximum cardinalities
- Four options for binary relationship types: **1:1, 1:N, N:1, N:M**.
  - Examples:



**N:M**

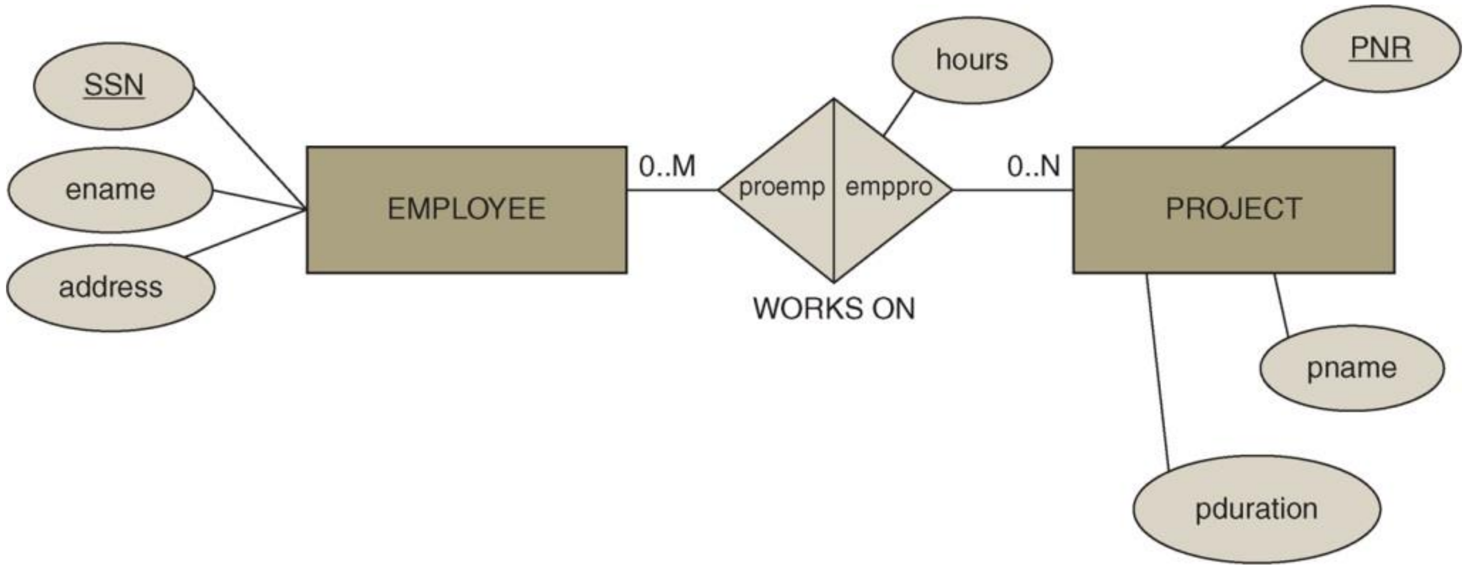
**1:1**

**1:N**

## Relationship Attribute Types

- **Relationship type can also have attribute types.**
  - E.g., The *hours* represents the number of hours an employee worked on a project. Its value is uniquely determined by a combination of an employee entity and a project entity.

    So, the hours is the attribute type of relationship WORKS ON.



## Outline

- Phases of Database Design
- Entity Relationship (ER) Model
  - Entity Types
  - Attribute Types
  - Relationship Types
  - ☞ **Weak Entity Types**
  - Ternary Relationship Types
  - Limitations of the ER Model
- Enhanced Entity Relationship (EER) Model
- UML Class Diagram
- Case Studies

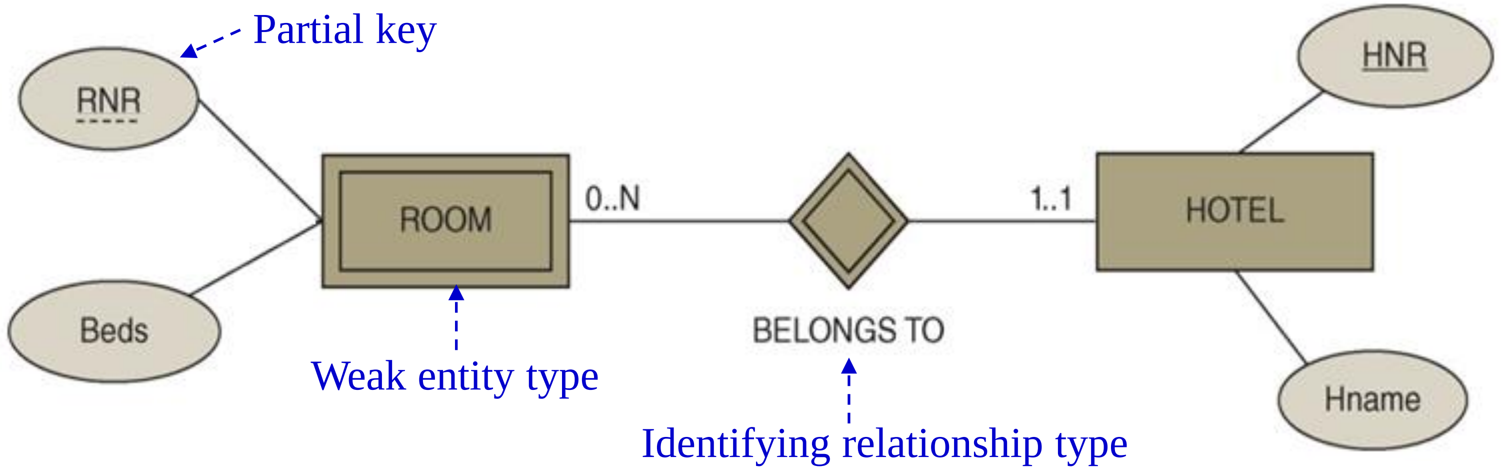
## Different Entity Types

- A **strong entity type** is an entity type that has a key attribute type. A regular entity type is a strong entity type.
- A **weak entity type** is an entity type that does not have a key attribute type of its own
  - The weak entity types are represented using a *double rectangles* in the ER model

## Weak Entity Types

- A **weak entity type** should be connected to its **owner entity type** from which it borrows an attribute type to make up a (composite) key attribute type with its own attribute type(s) (called **partial key attribute type**)
- All weak entities should participate to a relationship type which associates to their owner entity (not vice versa!) (**total participation**)
- The relationship type between a weak entity type and its owner entity type is called **identifying relationship type**.
  - The identifying relationship types are represented using a *double rhombus* symbol in the ER model

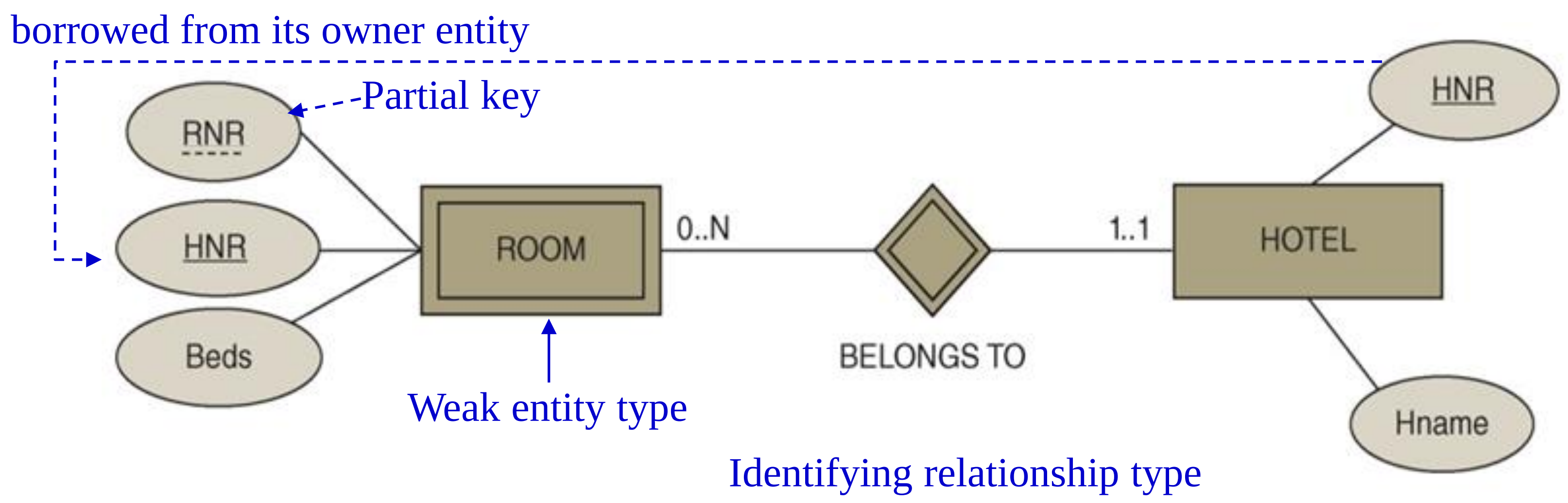
# Example of Weak Entity Type



**< Figure A >**

- Within a particular hotel, each room has a unique number but the same room number can occur for multiple rooms in different hotels. Hence, RNR does not suffice as a key attribute type.
- The ROOM entity type is a weak entity type since it cannot produce its own key attribute type.
- **NOTE** that you may consider an artificial (surrogate) key attribute type (e.g., id) for the weak entity, but it is not used for conceptual data model which is implementation independent.

## Example (cont.)



**< Figure B >**

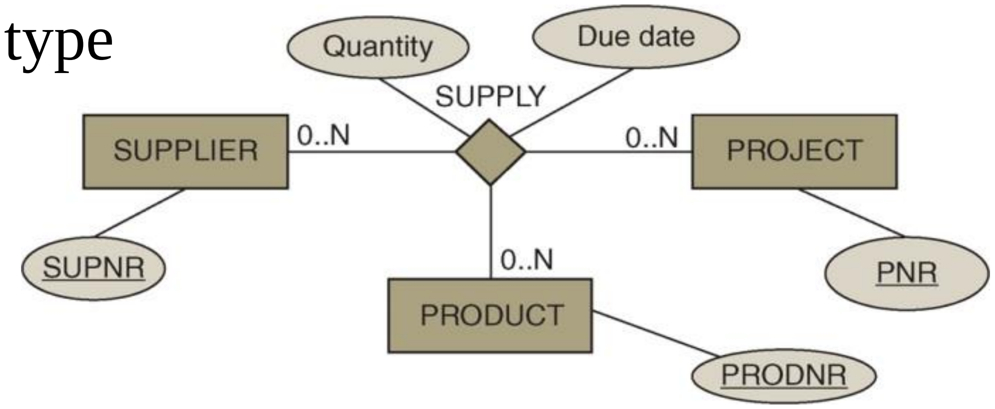
- Eventually the ROOM entity needs to borrow HNR from HOTEL to come up with a key attribute type which is now a combination of its partial key RNR and HNR.
- Both <Figure A> and <Figure B> are fine for representing the entity type, its attribute types, partial key attribute and key attribute.
- Often only partial key attribute(s) are presented in a weak entity like <Figure A>. In mapping the weak entity type to a logical schema (e.g., ROOM table), the partial key attribute type and its owner's primary key attribute type are used as the composite key of the logical schema (e.g., ROOM table).

## Outline

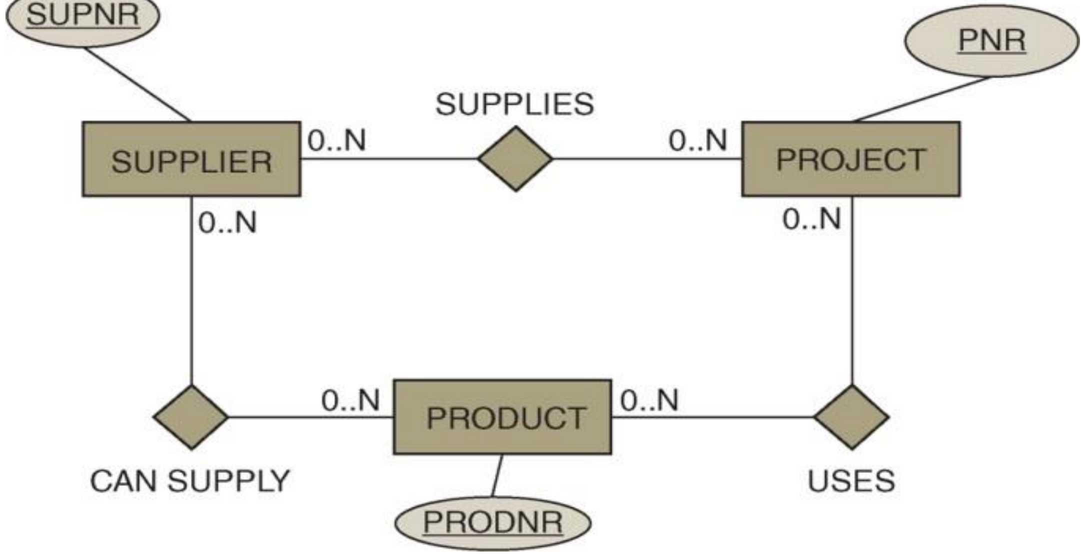
- Phases of Database Design
- Entity Relationship (ER) Model
  - Entity Types
  - Attribute Types
  - Relationship Types
  - Weak Entity Types
  - ☞ **Ternary Relationship Types**
  - Limitations of the ER Model
- Enhanced Entity Relationship (EER) Model
- UML Class Diagram
- Case Studies

# Ternary Relationship vs Binary Relationship Types

- A ternary relationship type



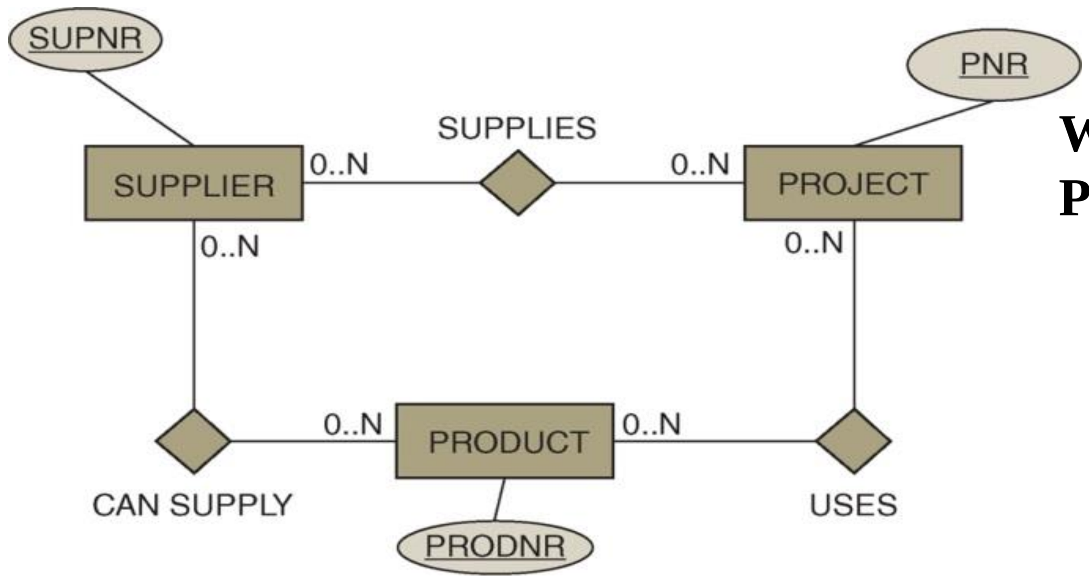
- Binary relationship types



## Case Example

- Assume that we have a situation where suppliers can supply products for projects.
  - A supplier can supply a particular product for multiple projects.
  - A product for a particular project can be supplied by multiple suppliers.
  - A project can have a particular supplier who supplies multiple products.
  - The model must also include the quantity and due date for supplying a particular product to a particular project by a particular supplier.
- Say we have two projects:
  - Project 1 uses a pencil and a pen, and Project 2 uses a pen.
  - Supplier Peters supplies the pencil for Project 1 and the pen for Project 2
  - Supplier Johnson supplies the pen for Project 1.
- Design an ERD <u>to find who supplies a certain product for a particular project</u>.

## Loss of Semantics with multiple Binary Relationships



**Who supplies the pen for Project 1?**

- From three binary relationship types, it is not clear who supplies the pen for project 1.
- This schema has **loss of semantics!**

Tables for three *n*:*n* binary relationships and their records:

SUPPLIES

| Supplier | Project |
|---|---|
| Peters | Project 1 |
| Peters | Project 2 |
| Johnson | Project 1 |

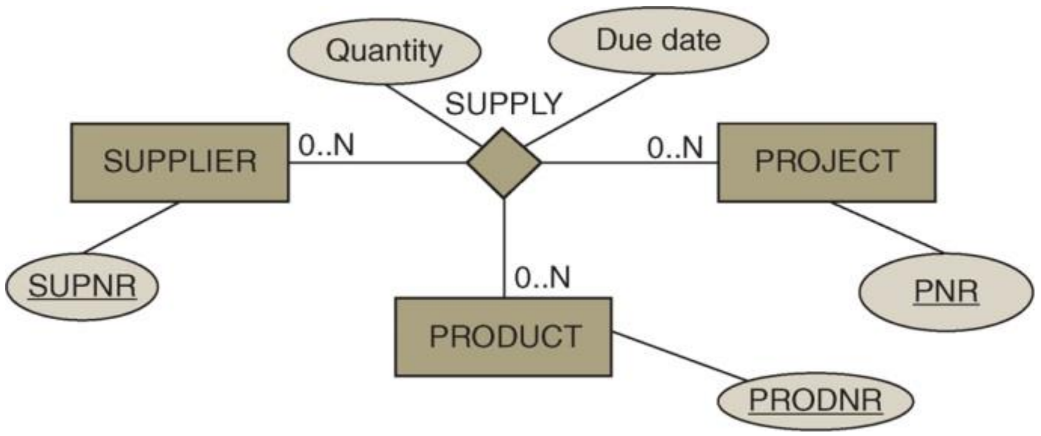
USES

| Product | Project |
|---|---|
| Pencil | Project 1 |
| Pen | Project 1 |
| Pen | Project 2 |

CAN SUPPLY

| Supplier | Product |
|---|---|
| Peters | Pencil |
| Peters | Pen |
| Johnson | Pen |

# Semantics with a Ternary Relationship Type

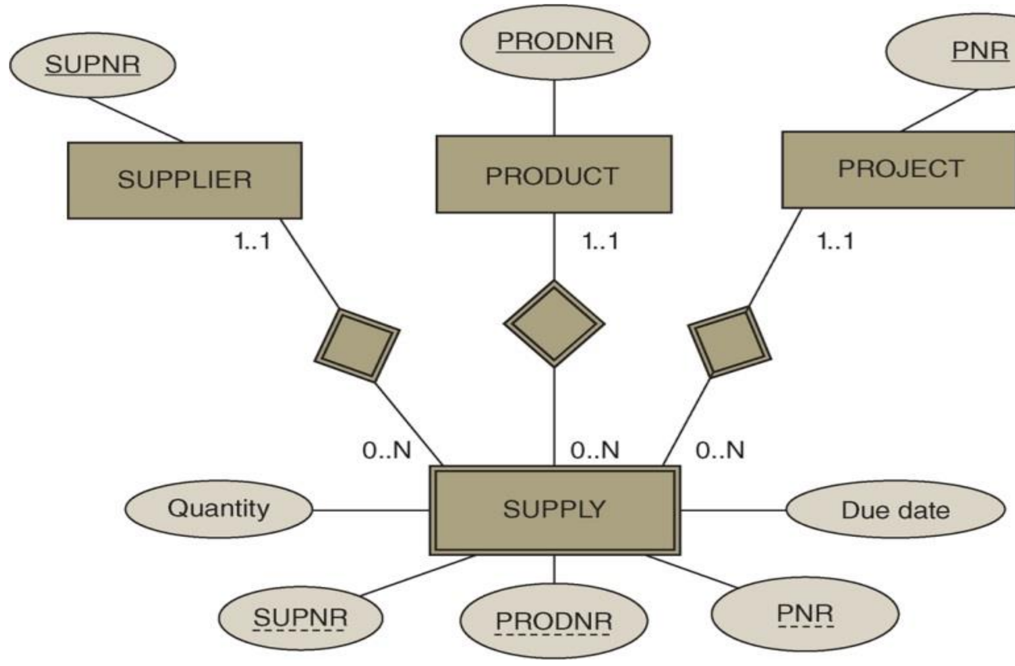


One ternary relationship table: **SUPPLY**

| **Supplier** | **Product** | **Project** | **Quantity** | **Due Date** |
|---|---|---|---|---|
| Peter | Pencil | Project1 | 200 | Jan-17-2020 |
| Peters | Pen | Project2 | 100 | Feb-17-2020 |
| Johnson | Pen | Project1 | 100 | Feb-21-2020 |

We know Johnson supplies the pen for project 1!!

## Model a Ternary Relationship using a Weak Entity and three Binary Identifying Relationships



The Supply weak entity table would be:

**SUPPLY**

| **<u>Supplier</u>** | **<u>Product</u>** | **<u>Project</u>** | **Quantity** | **Due Date** |
|---|---|---|---|---|
| Peter | Pencil | Project1 | 200 | Jan-17-2020 |
| Peters | Pen | Project2 | 100 | Feb-17-2020 |
| Johnson | Pen | Project1 | 100 | Feb-21-2020 |

## Outline

- Phases of Database Design
- Entity Relationship (ER) Model
  - Entity Types
  - Attribute Types
  - Relationship Types
  - Weak Entity Types
  - Ternary Relationship Types
  - ☞ **Limitations of the ER Model**
- Enhanced Entity Relationship (EER) Model
- UML Class Diagram
- Case Studies

## Limitations of the ER model

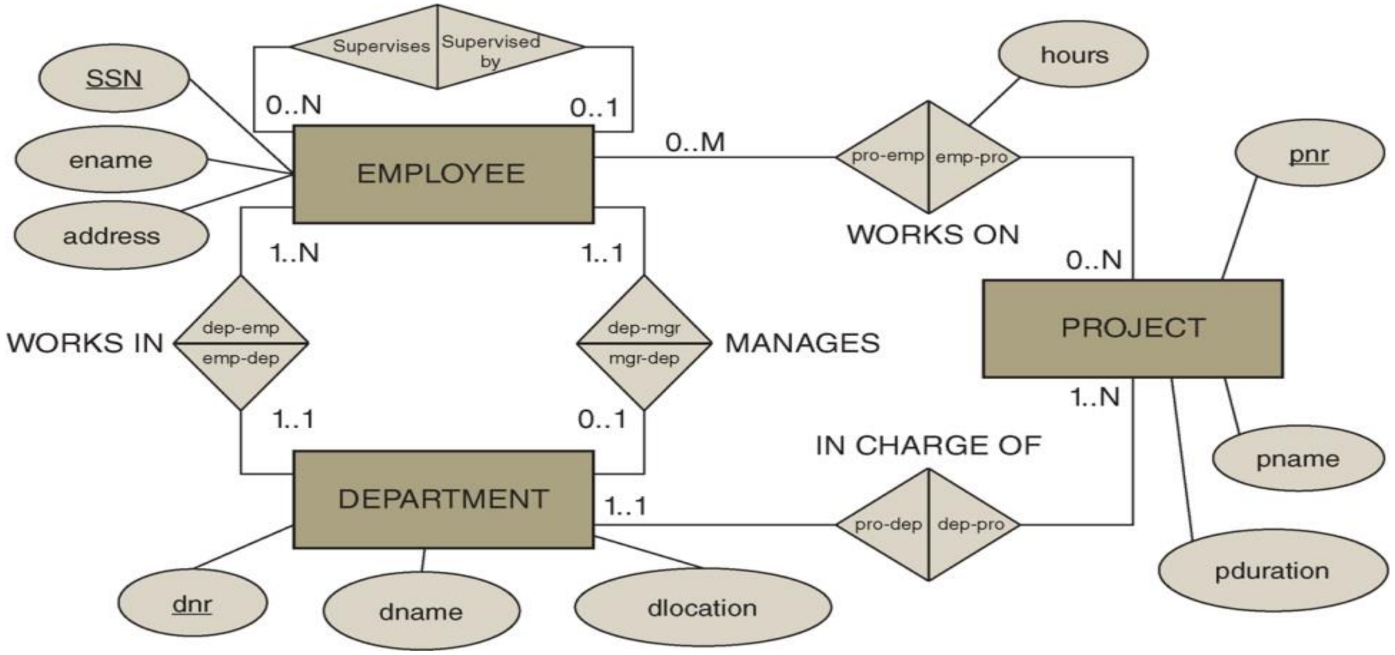
- ER model presents a temporary snapshot and **cannot model temporal constraints** which are constraints spanning a particular time interval
  - Examples
    - A project needs to be assigned to a department <u>after one month</u>
    - An employee cannot return to a department of which he <u>previously</u> was a manager
    - An employee needs to be assigned to a department <u>after six months,</u>
    - A purchase order must be assigned to a supplier <u>after two weeks</u>.
  - These rules need to be documented and followed up with application code.

## Limitations of the ER model

- ER model **cannot guarantee the consistency across multiple relationship types**
  - Examples of business rules that cannot be enforced in the ER model
    - Suppliers can only be assigned to purchase orders for products they can supply
    - An employee should work in the department that he/she manages
    - Employees should work on projects assigned to departments to which the employees belong.
  - Business rules with this requirement need to be documented and followed up with application code.

## Example

- Consider whether the ERD below can support the following cases:
  - An employee should work in the department that he/she manages
  - Employees should work on projects assigned to departments to which the employees belong.



## Limitations of the ER model (cont.)

- **Domains are not included** in the ER model
  - Examples
    - Hours should be positive
    - Prodtype must be red, white or sparkling,
    - Supstatus is an integer between 0 and 100
- **Functions are not included** in the ER model
  - Examples
    - Calculate average number of projects an employee works on.
    - Determine which supplier charges the maximum price for a product

## Outline

- Phases of Database Design
- Entity Relationship (ER) Model
- ☞ **Enhanced Entity Relationship (EER) Model**
  - Specialization / Generalization
  - Categorization
  - Aggregation
  - Designing the EER Model
- UML Class Diagram
- Case Studies

## Enhanced Entity Relationship (EER) Model

- **Enhanced Entity Relationship (EER) model** is an extension of the ER model.
- The EER model includes all the modeling concepts (entity types, attribute types, relationships types) of the ER model, and also three new additional semantic data modeling concepts: **specialization/generalization**, **categorization** and **aggregation**.

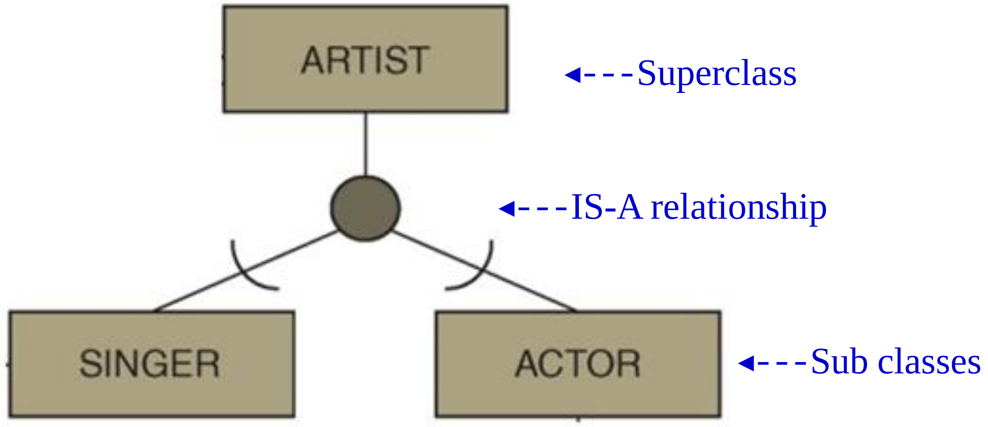
## Specialization/Generalization

- **Specialization** refers to the process of defining a set of subclasses (sub entity types) of an entity type
  - E.g., ARTIST superclass with subclasses SINGER and ACTOR
- Specialization corresponds to a **top-down process** of conceptual refinement
- A subclass <u>inherits all attribute types and relationship types from its superclass (super entity type)</u>
- The specialization can then establish <u>additional specific attribute types for each subclass</u>
  - E.g., singer can have a music style attribute type
- The specialization can also establish <u>additional specific relationship types for each subclass</u>
  - E.g., actor can act in movies, singer can be part of a band

## Specialization/Generalization

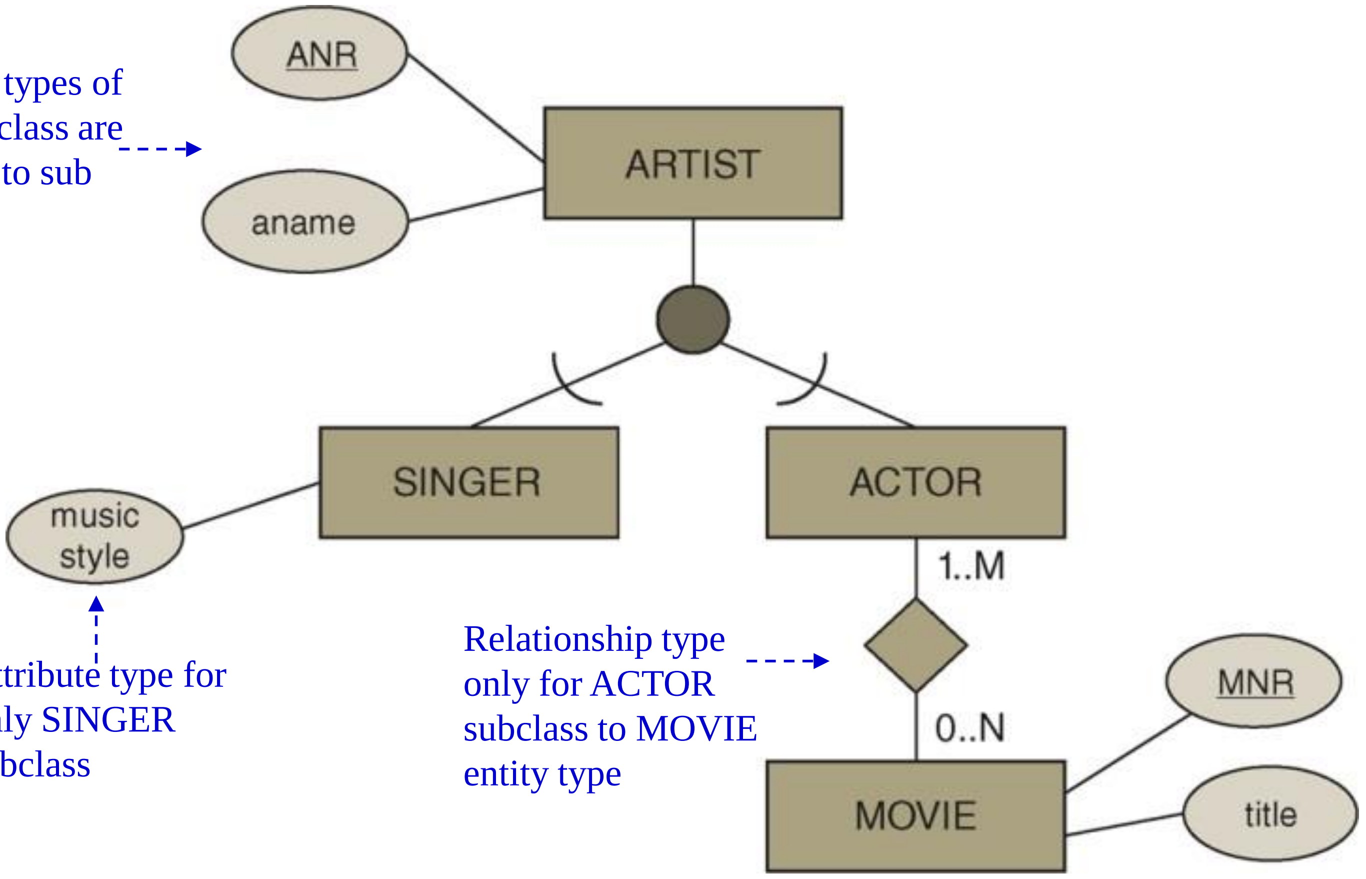
- **Generalization**, also called abstraction, is the reverse process of specialization
  - Generalization corresponds to a **bottom-up process** of conceptual synthesis

- **IS-A Relationship** is used to associate the superclass and its sub classes.



## Example of Specialization/Generalization

Attribute types of the superclass are inherited to sub classes.



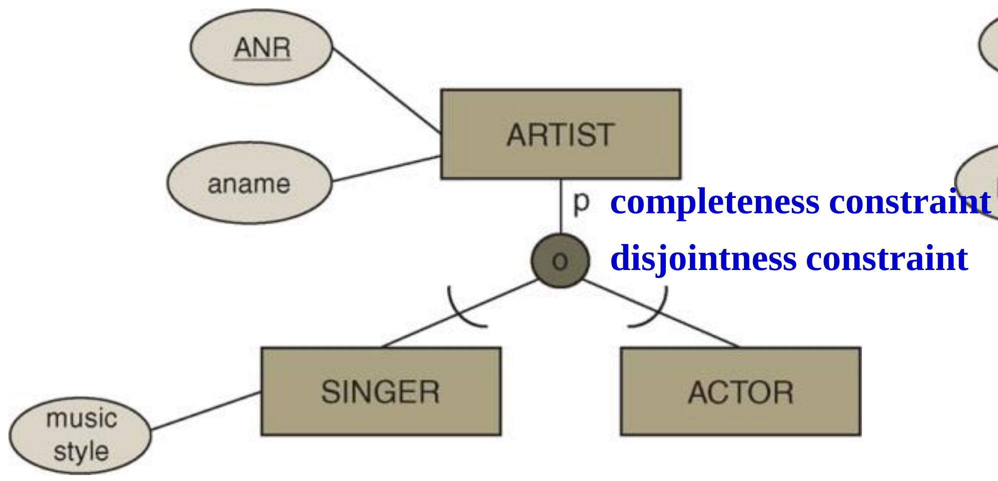
Attribute type for only SINGER subclass

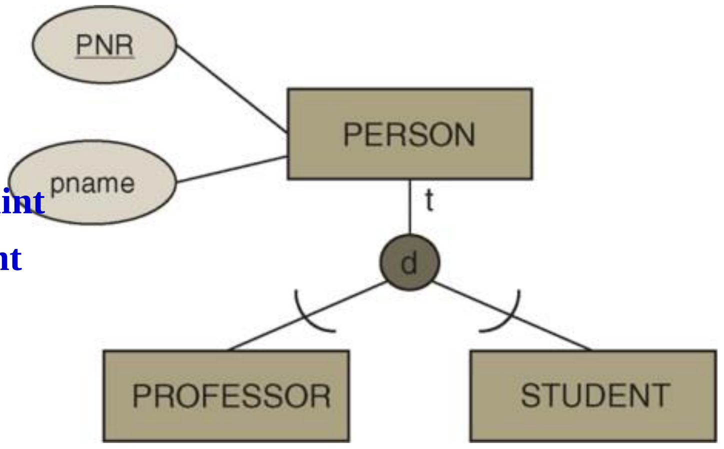
Relationship type only for ACTOR subclass to MOVIE entity type

## Constraints of IS-A Relationship

- The **disjointness constraint** of the IS-A relationship specifies to what subclasses an entity of the superclass can belong to
  - ***disjoint*** is a specialization whereby an entity can be a member of at most one of the subclasses
  - ***overlap*** is a specialization whereby the same entity may be a member of more than one subclass
- The **completeness constraint** indicates if all entities of the superclass should belong to one of the subclasses or not
  - ***total*** is a specialization whereby every entity in the superclass must be a member of some subclass
  - ***partial*** is a specialization which allows an entity to only belong to the superclass and to none of the subclasses

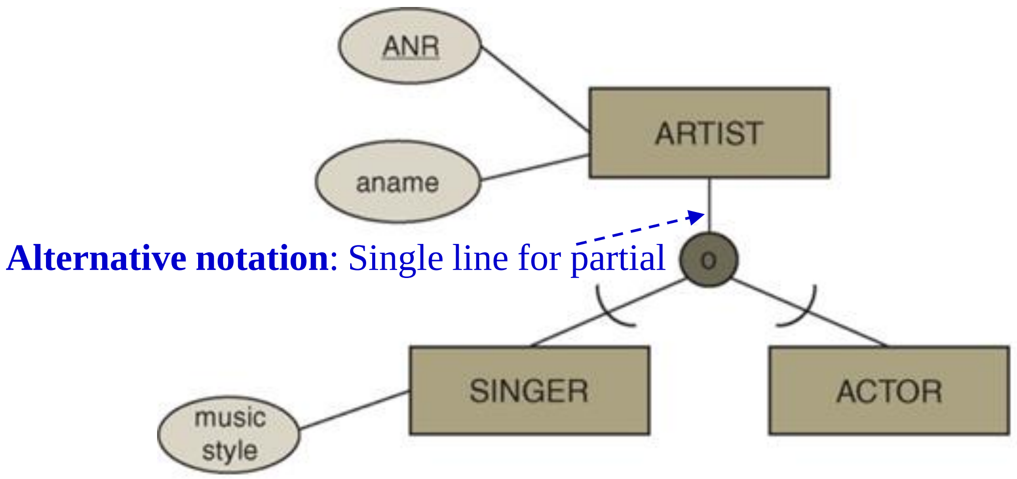
# IS-A Constraints in EER Model

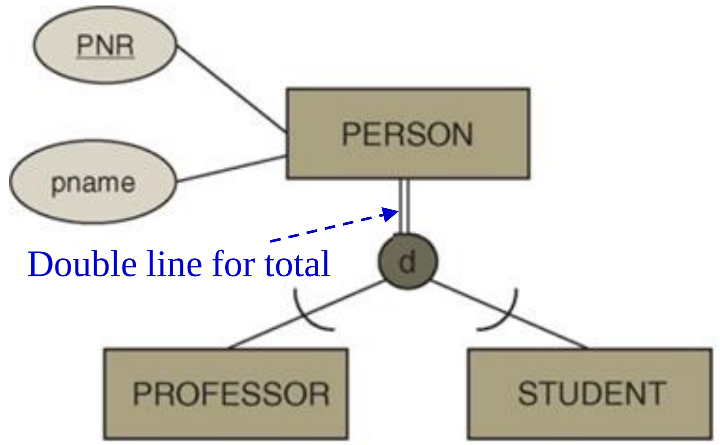




(a) Partial and with overlap

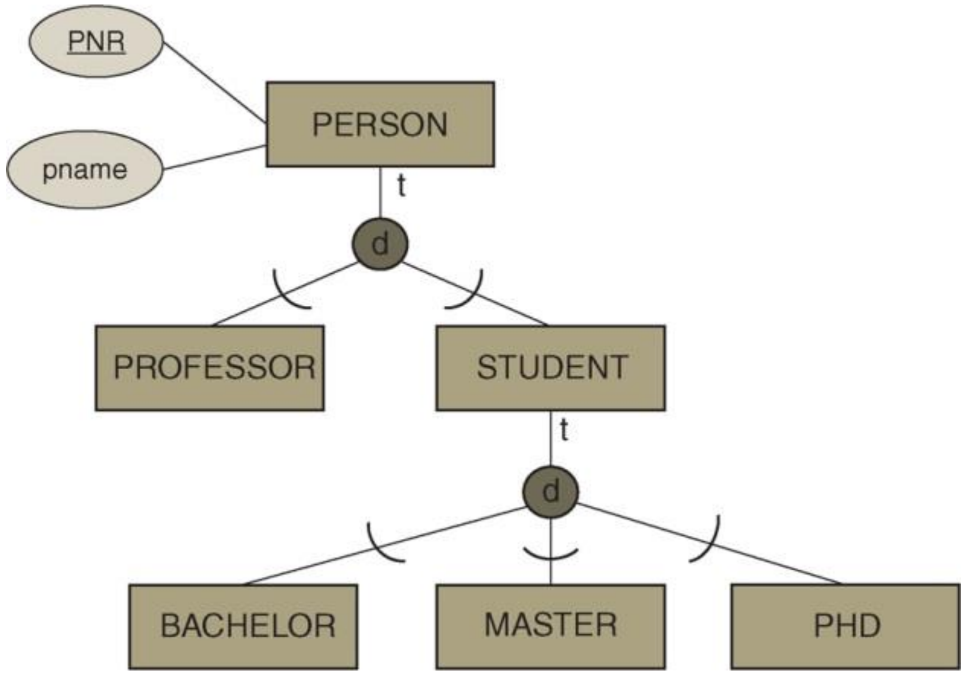
(b) Total and with disjoint





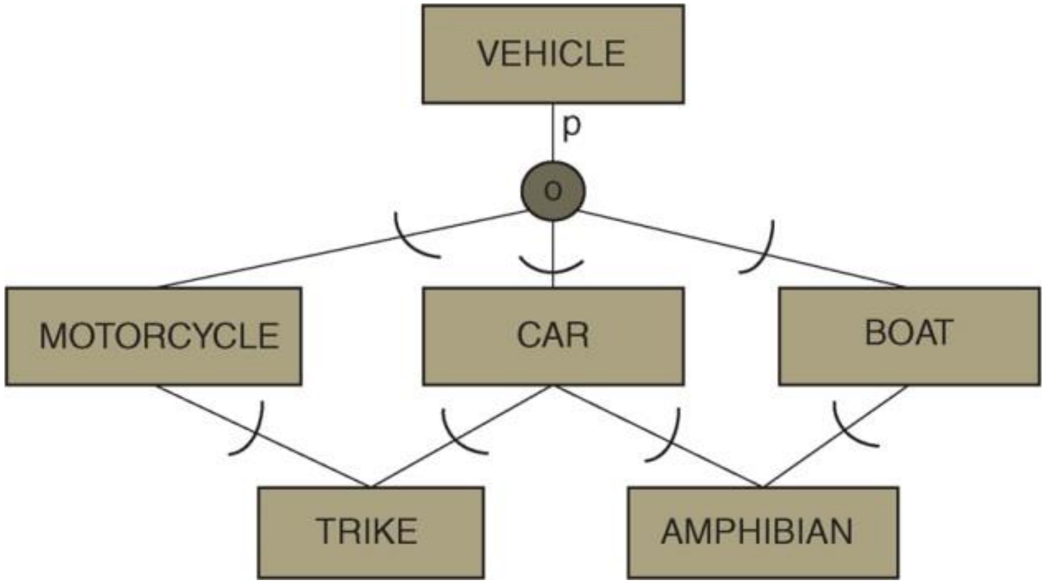
## Specialization Hierarchy

- A specialization can be several levels deep – A subclass can again be a superclass of another specialization.
- In a **specialization hierarchy**, every subclass can only have a single superclass and inherits the attribute types and relationship types of all its predecessor superclasses all the way up to the root of the hierarchy



## Specialization Lattice

- In a **specialization lattice**, a subclass can have multiple superclasses (multiple inheritance)

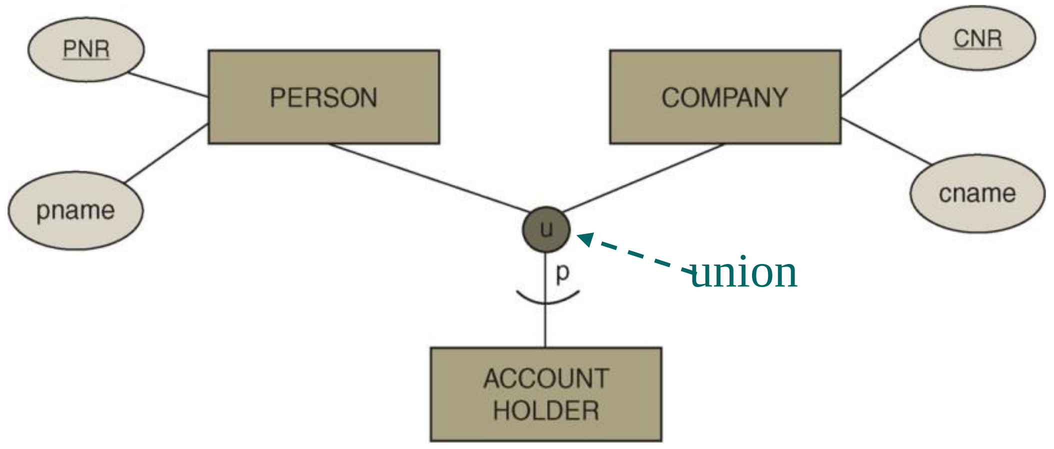


## Outline

- Phases of Database Design
- Entity Relationship (ER) Model
- Enhanced Entity Relationship (EER) Model
  - Specialization / Generalization
  - ☞ **Categorization**
  - Aggregation
  - Designing the EER Model
- UML Class Diagram
- Case Studies

## Categorization

- A **category** is a subclass that has several possible superclasses
- Each superclass represents a different entity type
- The category represents <u>a collection of entities that is a subset of the union of the superclasses</u>

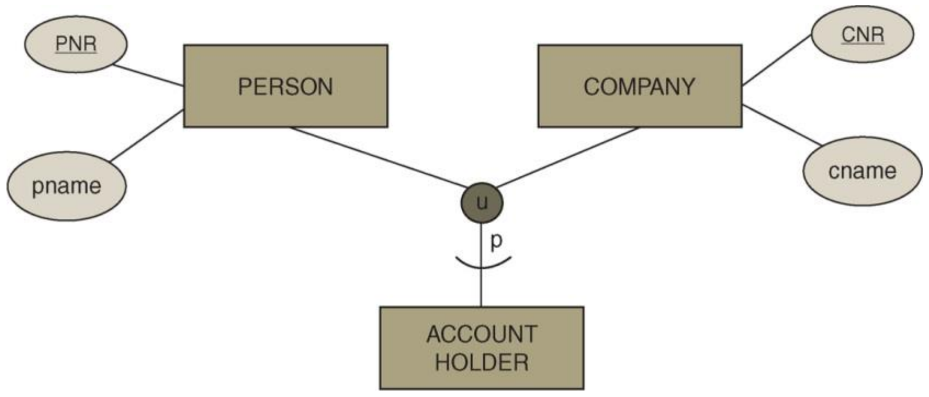


  - The superclasses PERSON and COMPANY have been categorized into an ACCOUNT HOLDER subclass
  - The account holder entities are a subset of the union of the person and company entities

## Categorization (cont.)

- **Inheritance** in the case of categorization corresponds to an entity inheriting only the attributes and relationships of that superclass it is a member of (**selective inheritance**)
  - E.g., Some account holders inherit their attributes and relationships from person, whereas others inherit them from company
- A categorization can be *total* or *partial*
  - ***Total*** **categorization**: all entities of the superclasses belong to the subclass.
  - ***Partial*** **categorization** : not all entities of the superclasses belong to the subclass

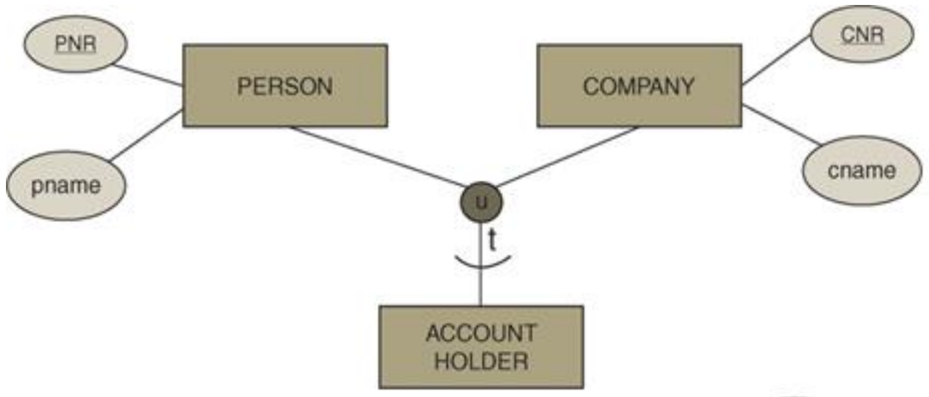
# Example

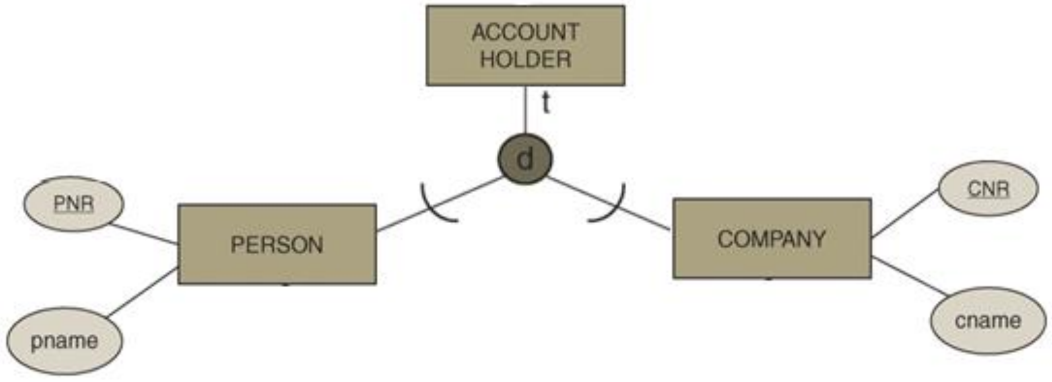


- **Partial categorization**: Not all persons or companies are account holders

- **Total categorization**:

All person and company entities are also acount holders.





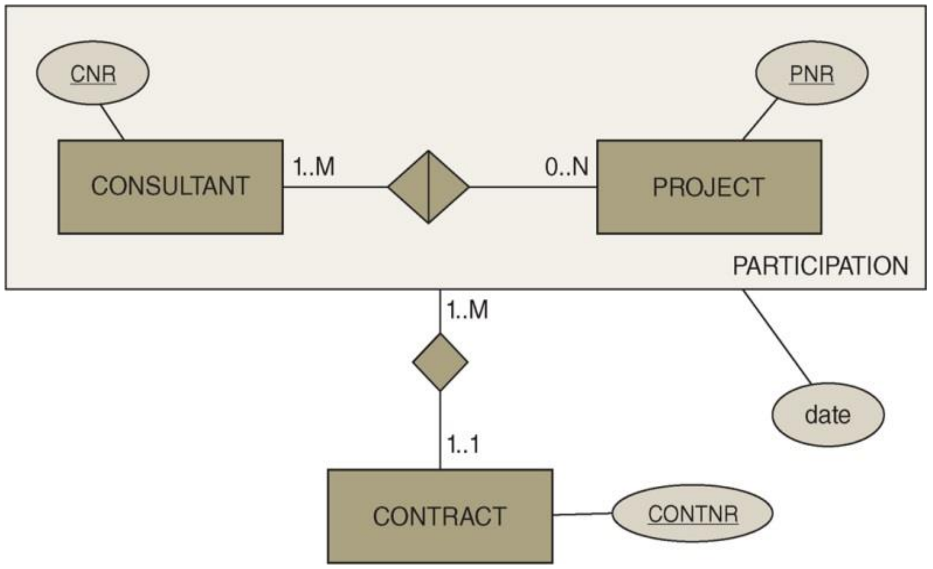
Total categorization can also be represented as a specialization.

## Outline

- Phases of Database Design
- Entity Relationship (ER) Model
- Enhanced Entity Relationship (EER) Model
  - Specialization / Generalization
  - Categorization
  - ☞ **Aggregation**
  - Designing the EER Model
- UML Class Diagram
- Case Studies

# Aggregation

- Entity types that are related by a particular relationship type can be combined or aggregated into **a higher-level aggregate entity type**
- **Aggregation** is especially useful when the aggregate entity type has its own attribute types and/or relationship types



- Both entity types and the corresponding relationship type is aggregated into the aggregate concept PARTICIPATION.
- The date attribute represents the date at which a consultant started working on a project.
- The aggregate also participates in a relationship type with CONTRACT

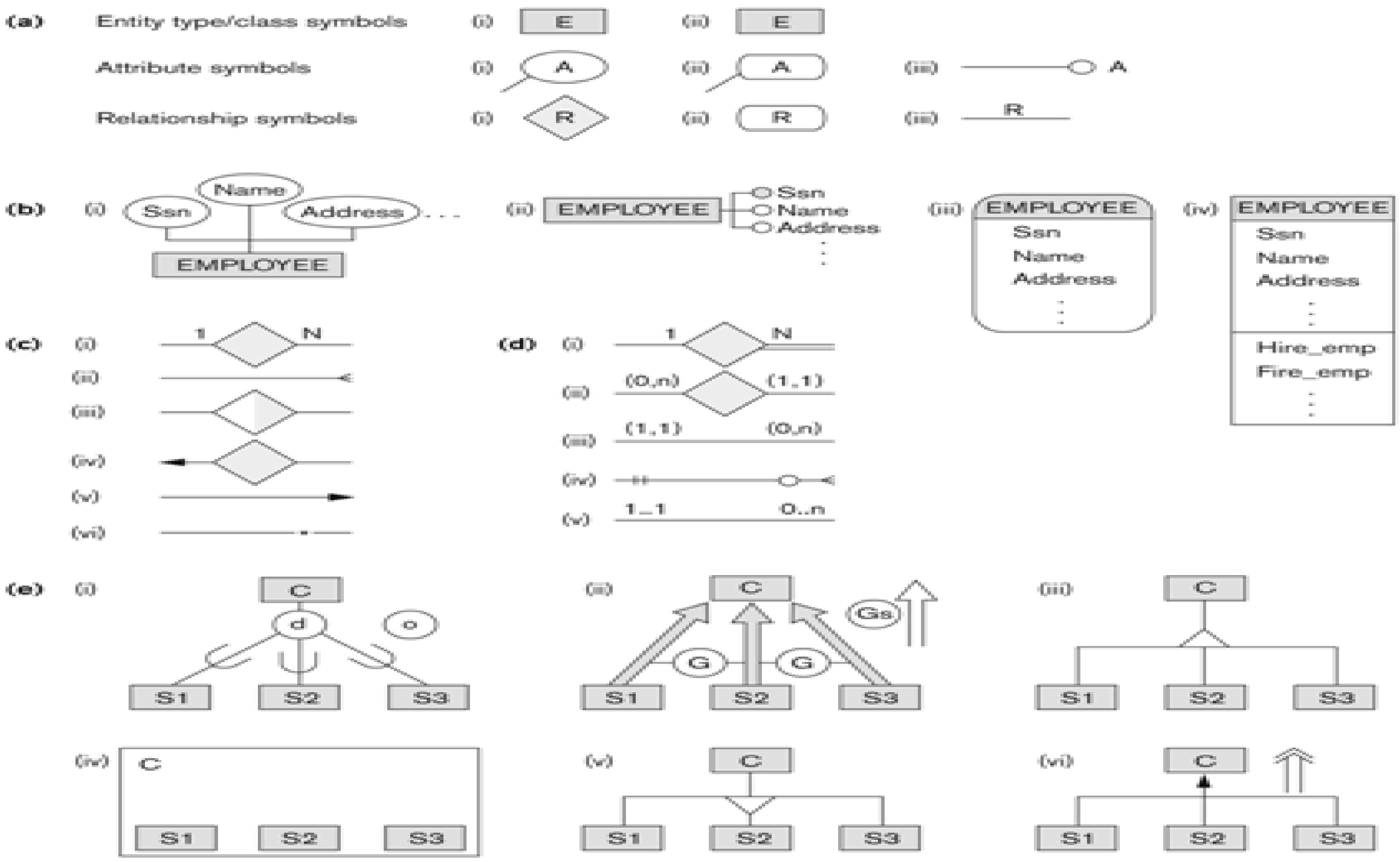
## Outline

- Phases of Database Design
- Entity Relationship (ER) Model
- Enhanced Entity Relationship (EER) Model
  - Specialization / Generalization
  - Categorization
  - Aggregation
  - ☞ **Designing the EER Model**
- UML Class Diagram
- Case Studies

## Designing the EER Model

1. Identify the **entity types**
2. Identify the **relationship types** and assert their **degree**
3. Assert the **cardinality ratios** (1 vs. N) and **participation constraints** (total vs. partial participation)
4. Identify the **attribute types** and assert whether they are simple or composite, single or multiple valued, derived or not
5. Link each attribute type to an entity type or a relationship type
6. Denote the **key attribute type(s)** of each entity type
7. Identify the **weak entity types** and **their partial keys**
8. Apply abstractions such as **generalization/specialization, categorization** and **aggregation**
9. Assert the characteristics of each abstraction such as **disjoint or overlapping, total or partial**

# Alternative ERD Notations



(a) Entity type/class symbols (i) E (ii) E
Attribute symbols (i) A (ii) A (iii) A
Relationship symbols (i) R (ii) R (iii) R
(b) (i) Name Ssn Address EMPLOYEE
(ii) EMPLOYEE Ssn Name Address
(iii) EMPLOYEE Ssn Name Address
(iv) EMPLOYEE Ssn Name Address Hire_emp Fire_emp
(c) (i) 1 N (ii) (iii) (iv) (v) (vi)
(d) (i) 1 N
(ii) (0,n) (1,1)
(iii) (1,1) (0,n)
(iv)
(v) 1..1 0..n
(e) (i) C d o S1 S2 S3
(ii) C Gs G G S1 S2 S3
(iii) C S1 S2 S3
(iv) C S1 S2 S3
(v) C S1 S2 S3
(vi) C S1 S2 S3

Alternative notations. (a) Symbols for entity type/class, attribute, and relationship. (b) Displaying attributes. (c) Displaying cardinality ratios. (d) Various (min, max) notations. (e) Notations for displaying specialization/generalization.

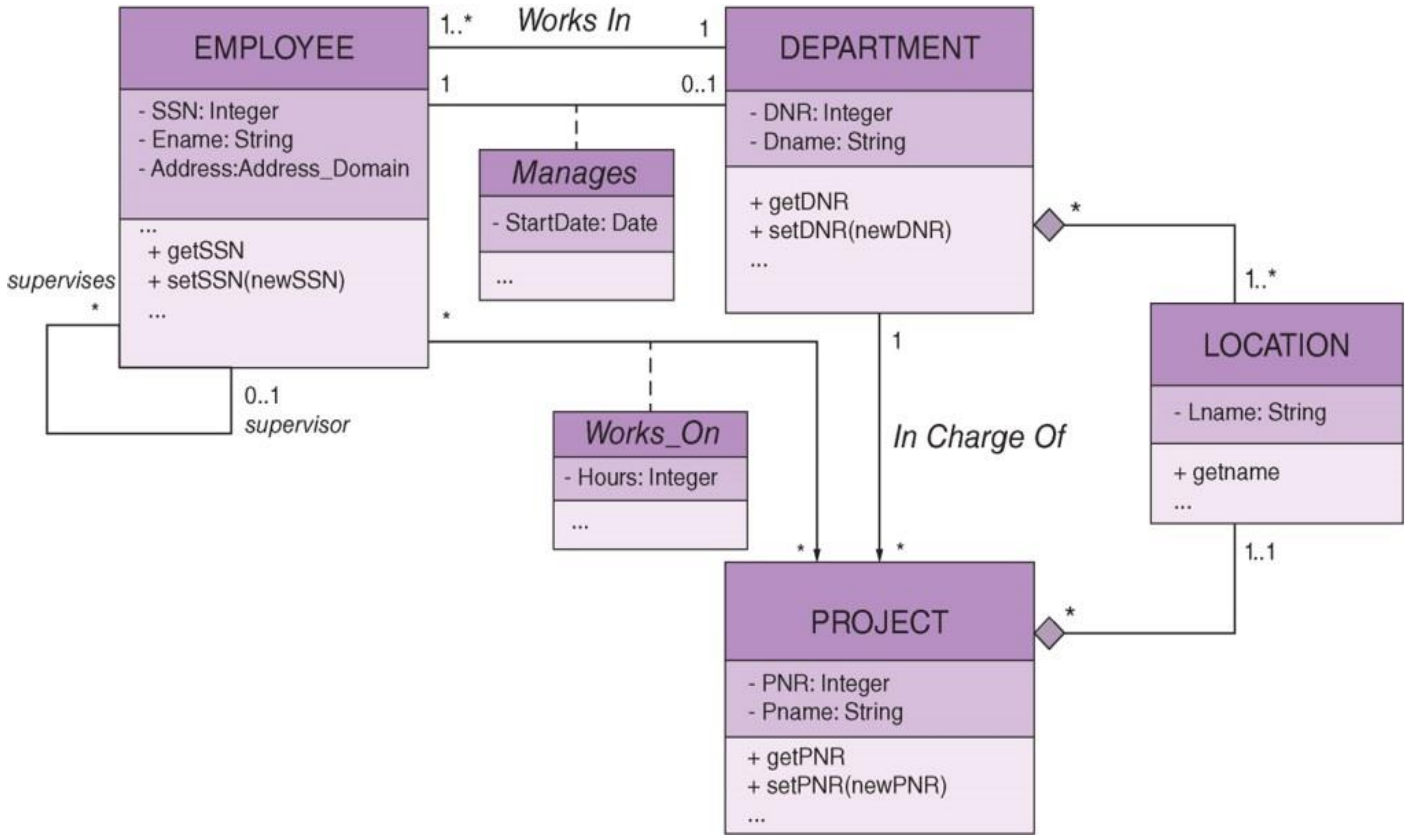
## Outline

- Phases of Database Design
- Entity Relationship (ER) Model
- Enhanced Entity Relationship (EER) Model
- **UML Class Diagram**
  - Origin, Recap of Object Orientation
  - Classes, Variables, Access Modifiers
  - Associations
  - Specialization / Generalization
  - Aggregation
  - Advanced UML Modeling Concepts
  - UML versus EER
- Case Studies

## Origin

- The **Unified Modeling Language (UML)** is a modeling language which assists in the specification, visualization, construction and documentation of artifacts of a software system
  - UML was accepted as a standard by the Object Management Group (OMG) in 1997 and approved as an ISO standard in 2005
  - Most recent version is UML 2.5, introduced in 2015
- UML offers various diagrams such as use case diagrams, sequence diagrams, package diagrams, deployment diagrams, etc.
- From a database modeling perspective, the **class diagram** is the most important
- **UML class diagrams** are also appeared for displaying ER concepts that is used in several commercial design tools

## UML Class Diagram Example



## Recap of Object Orientation

- A **class** is a blueprint definition for a set of objects
  - Compare to entity type in ER
- Conversely, an **object** is an instance of a class
  - Compare to entity in ER
- Object is characterized by both **variables** and **methods**
  - Variables correspond to attribute types and variable values to attributes in the ER
  - No ER equivalent for methods
- Example class **Student**
  - Example objects: Bart, Wilfried, Seppe
  - Example variables: student's name, gender and birthdate
  - Example methods: calcAge, isBirthday, hasPassed(courseID)

## Recap of Object Orientation

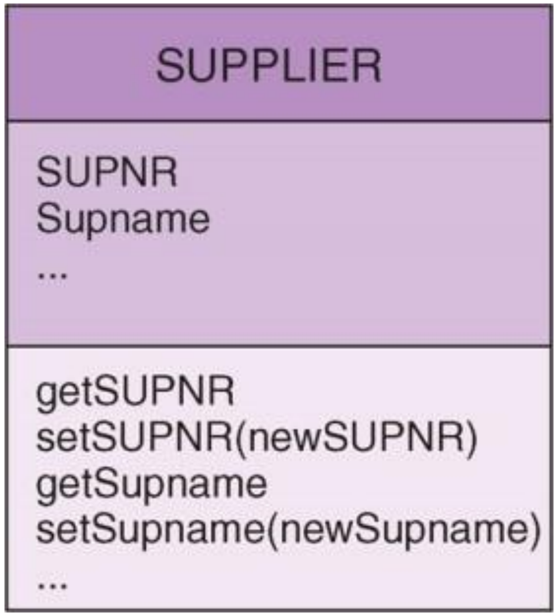
- **Information hiding** (a.k.a. **encapsulation**) states that the variables of an object can only be accessed through either getter or setter methods
  - A getter method is used to retrieve the value of a variable
  - A setter method assigns a value to the variable
- **Inheritance**
  - A superclass can have one or more subclasses which inherit both the variables and methods from the superclass
- **Method overloading**
  - Various methods in the same class can have the same name, but a different number or type of input arguments

## Outline

- Phases of Database Design
- Entity Relationship (ER) Model
- Enhanced Entity Relationship (EER) Model
- UML Class Diagram
  - Origin, Recap of Object Orientation
  - ☞ **Classes, Variables, Access Modifiers**
  - Associations
  - Specialization / Generalization
  - Aggregation
  - Advanced UML Modeling Concepts
  - UML versus EER
- Case Studies

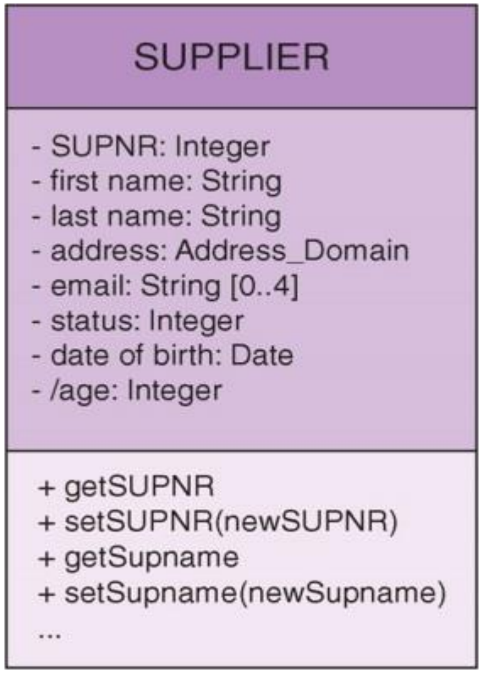
## Classes

- In a UML class diagram, a class is represented as a *rectangle* with three sections.
  - upper section – the name of the class
  - middle section – the variables
  - bottom section – the methods



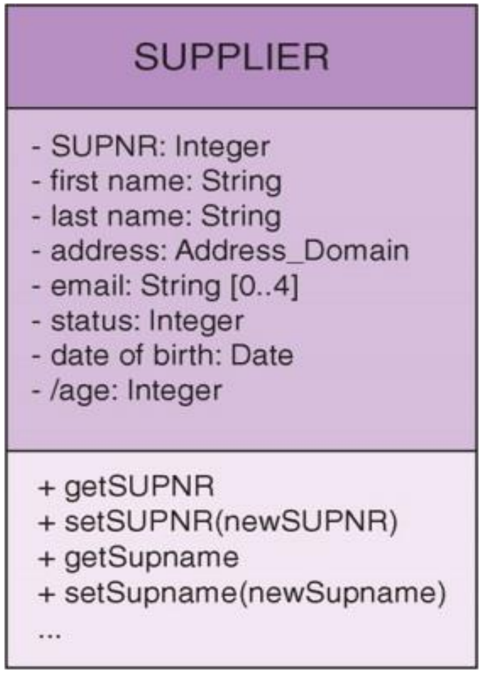
## Variables

- UML provides a set of primitive types such as String, Integer, and Boolean
- It is also possible to define **your own data types** or **domains**
- **Composite variables**
  - Option 1: Decompose them into their parts
    - e.g., first name, last name
  - Option 2: Create a new domain and use it
    - e.g., Name_Domain type for name
- **Multi-valued variables**
  - Indicate the multiplicity of the variable – How many values of the variable will be created when an object is instantiated.
    - e.g., email: String [0.. 4], email: String [*] (infinite number of emails)



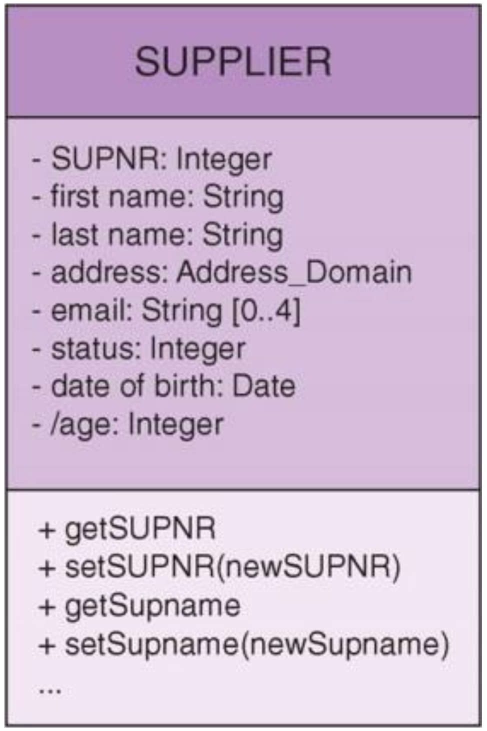
## Variables (cont.)

- **Derived variables**
  - Be preceded by a forward slash, e.g., age
- **Variables with unique values**
  - Variables with unique values (~ **key attribute types** in ER) are not directly supported in UML
    - A UML class diagram is assumed to be implemented using an OODMBS in which every object created is assigned a unique and immutable object identifier (OID) that it keeps during its entire lifetime
  - The OID can be used to uniquely identify objects and no other variables are needed to serve as a key



## Access Modifiers

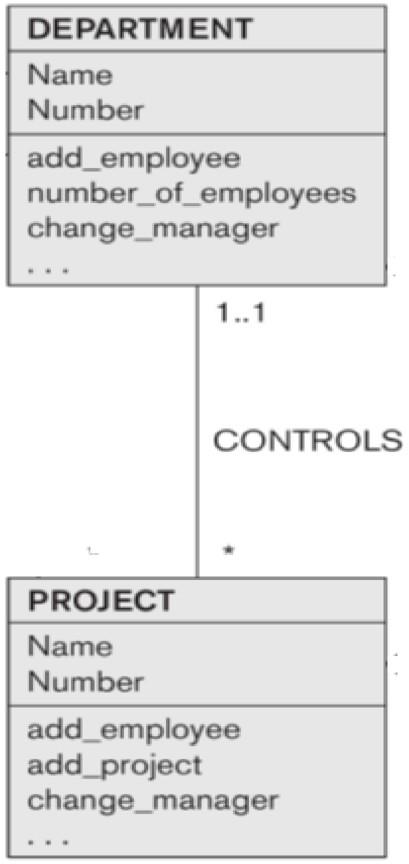
- **Access modifiers** can be used to specify who can have access to a variable or method
- Three types
  - **private** (denoted by '–'): variable or method can only be accessed by the class itself
  - **public** (denoted by '+'): variable or method can be accessed by any other class
  - **protected** (denoted by '#'): variable or method can be accessed by both the class and its subclasses
- It is recommended to declare all variables as private (information hiding)



## Associations

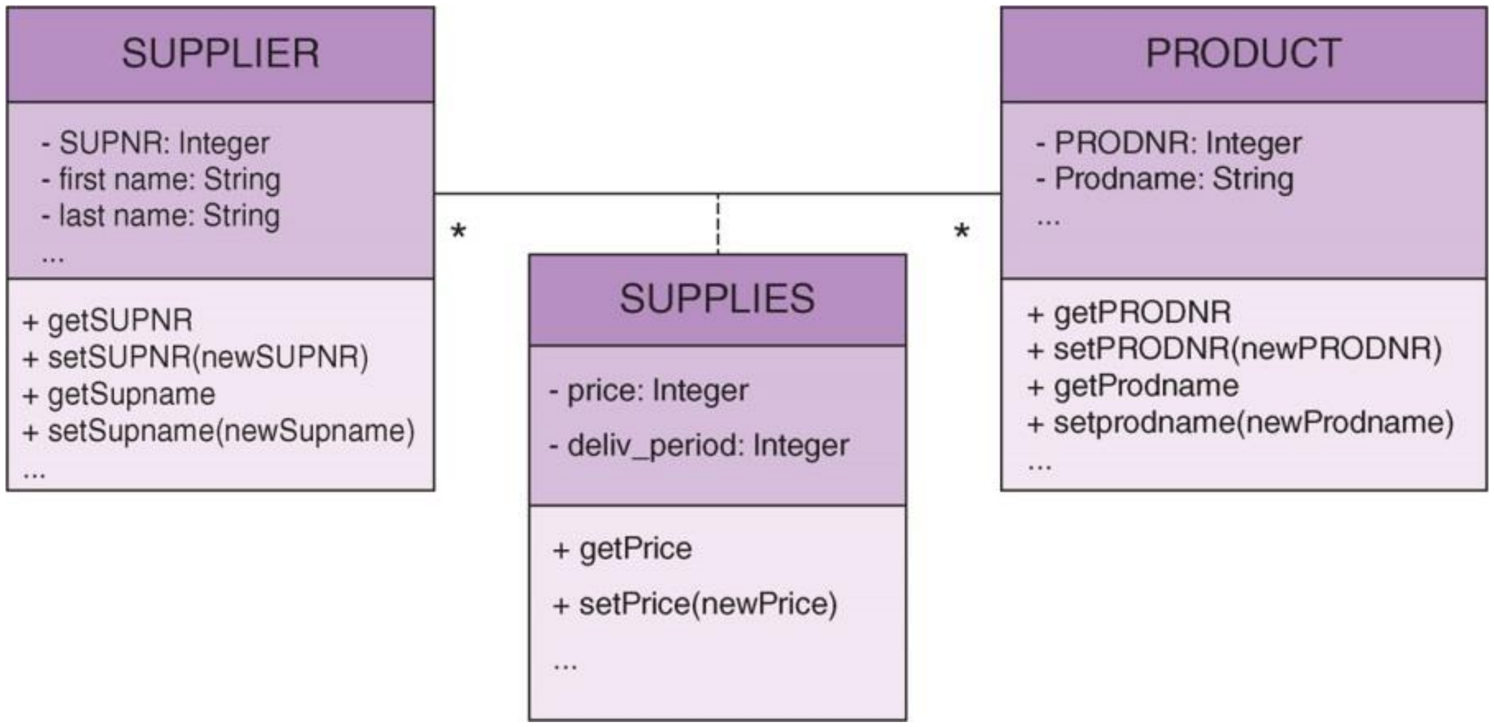
- An **association** corresponds to a **relationship type** in ER
- Multiple associations can be defined between the same classes
- A particular occurrence of an association is referred to as a **link**
- An association is characterized by its **multiplicities** (**cardinalities** in ER)

| UML class diagram **multiplicity** | ER diagram **cardinality** |
|---|---|
| * | 0..N |
| 0..1 | 0..1 |
| 1..* | 1..N |
| 1 | 1..1 |



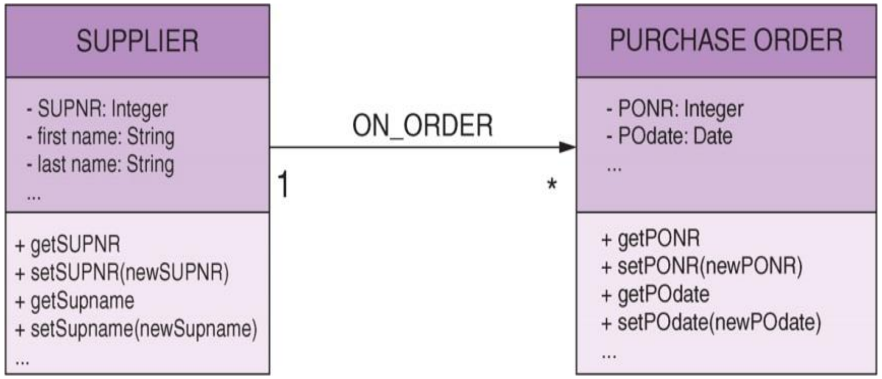
## Association Class

- In case an association has variables and/or methods on its own, it can be modelled as an **association class**



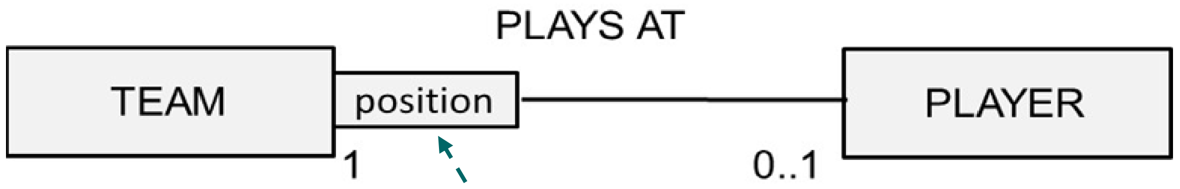
## Unidirectional vs. Bidirectional Association

- Associations can be augmented with direction reading **arrows**, which specify the direction of querying or navigating through it
  - In a **unidirectional association**, there is only a single way of navigating, as indicated by the arrow
  - In a **bidirectional association**, both directions are possible and hence there is no arrow.
- Example
  - All purchase orders can be retrieved through a supplier object.
  - It is not possible to navigate from a purchase order object to a supplier object.



## Qualified Association

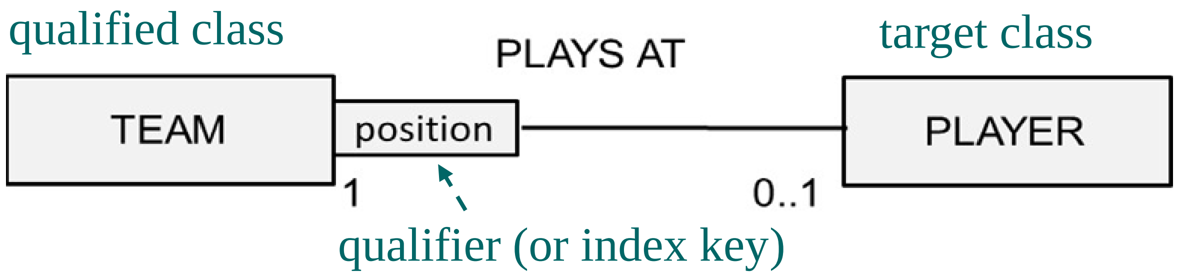
- A **qualified association** is a special type of binary association that uses a qualifier to further refine the association
- The **qualifier** specifies one or more variables that are used as key that are used to index a subset of relationship instances
  - Because of this extra key, the multiplicity of the association is reduced.
- A qualifier is visually represented as a rectangle attached to the qualified end of the association relationship.



qualifier (or index key)

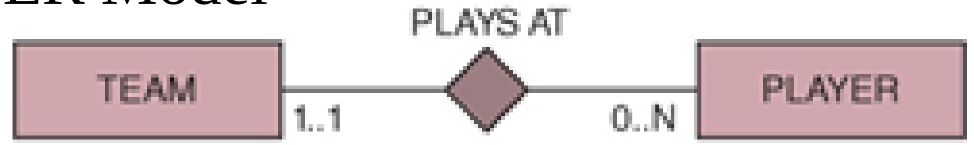
## Qualified Association (cont.)

- In a navigation context, qualifiers are used to select a specific object pair from the set of all related objects in that association.
- In an implementation context, each qualifier value points to a unique target object.
- Generally, if an application requires the retrieval of data based on search keys, the model should use qualified associations.



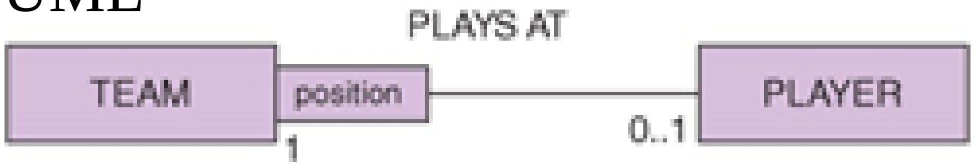
# Example of Qualified Association

- ER Model



  - 1:N relationship between TEAM and PLAYER
    - A team can have zero to N players. A player is always related to exactly one team

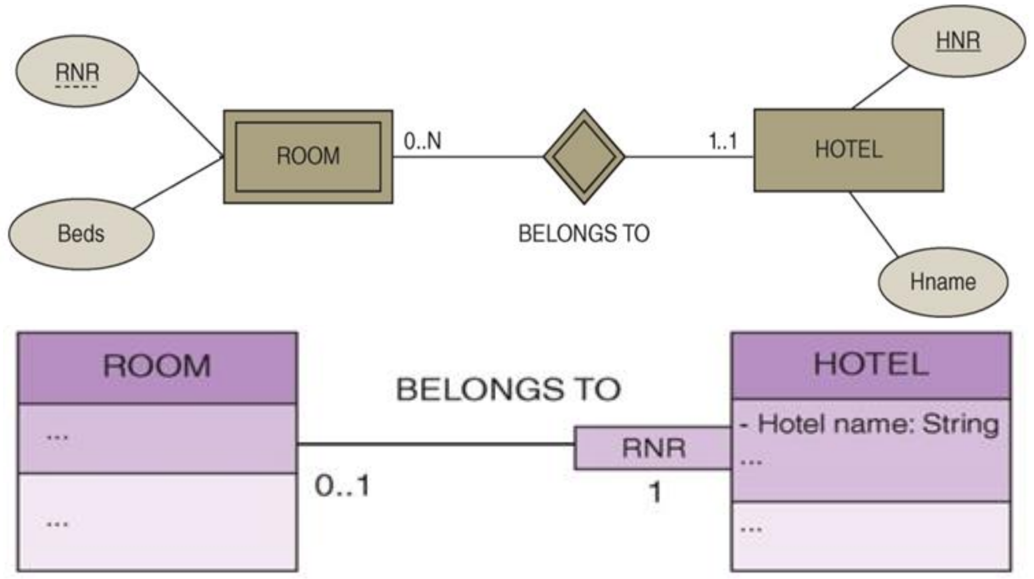
- UML



  - A team at a given position has zero or one players, whereas a player always belongs to exactly one team
  - Because of this qualifier, the multiplicity of the association is reduced.

## Qualified Association for Weak Entity Types

- Qualified associations can be used to represent weak entity types in ER.
- Example



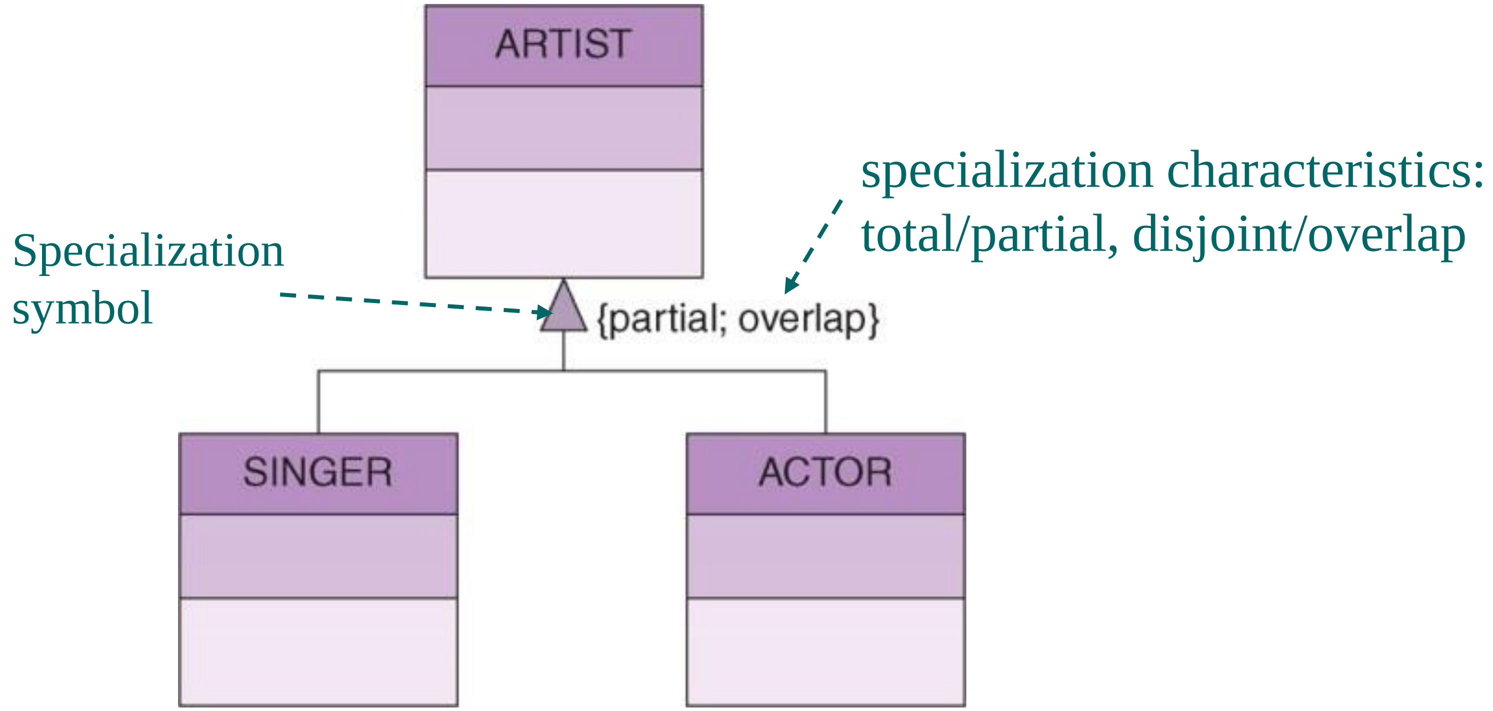
  - The room number is defined as a qualifier or index key.
  - A hotel combined with a given room number corresponds to zero or one room. A room always belongs to one hotel.

## Outline

- Phases of Database Design
- Entity Relationship (ER) Model
- Enhanced Entity Relationship (EER) Model
- UML Class Diagram
  - Origin, Recap of Object Orientation
  - Classes, Variables, Access Modifiers
  - Associations
  - ☞ **Specialization / Generalization**
  - Aggregation
  - Advanced UML Modeling Concepts
  - UML versus EER
- Case Studies

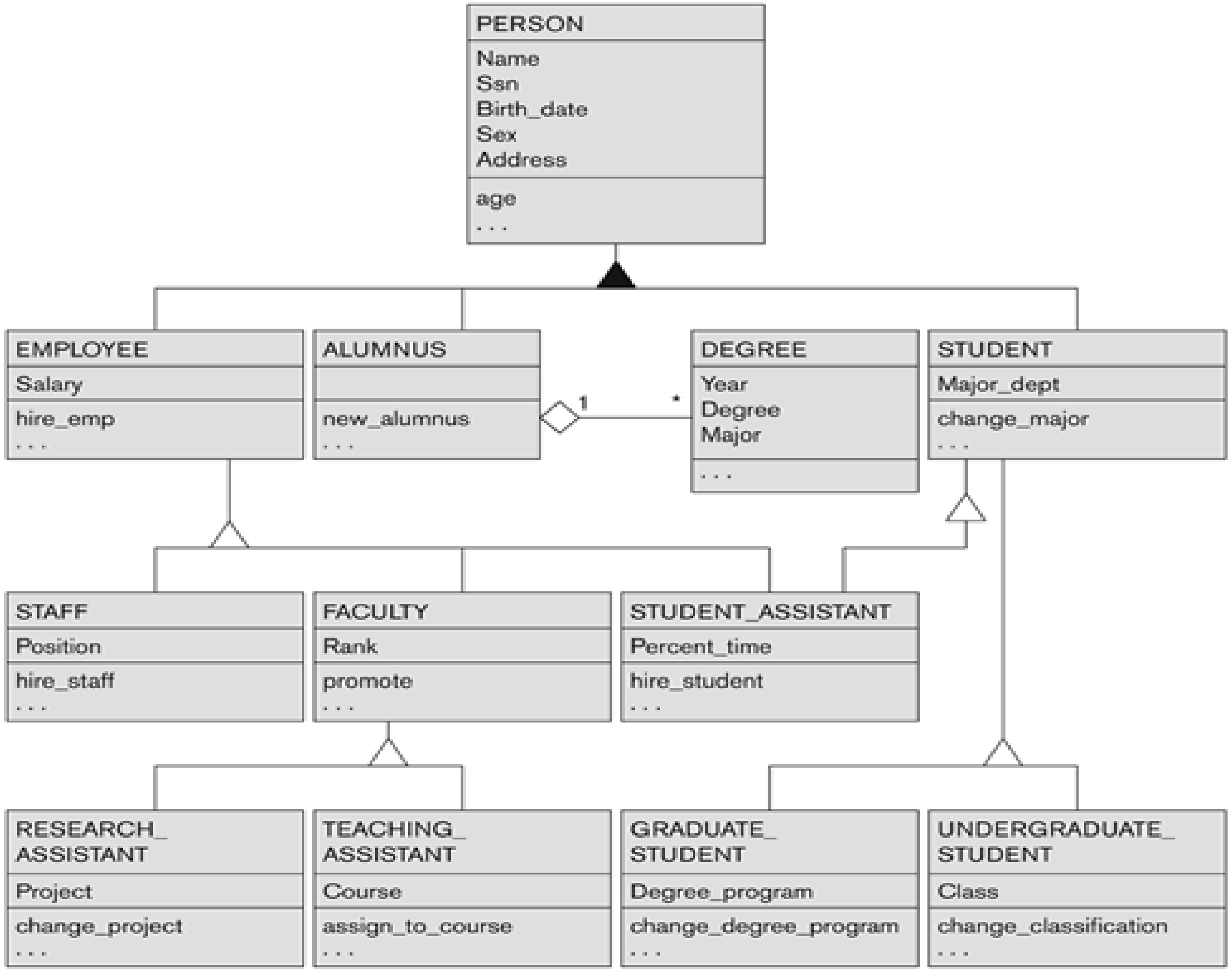
## Specialization / Generalization

- Similar to the EER model, UML also supports **specialization / generalization relationships**.



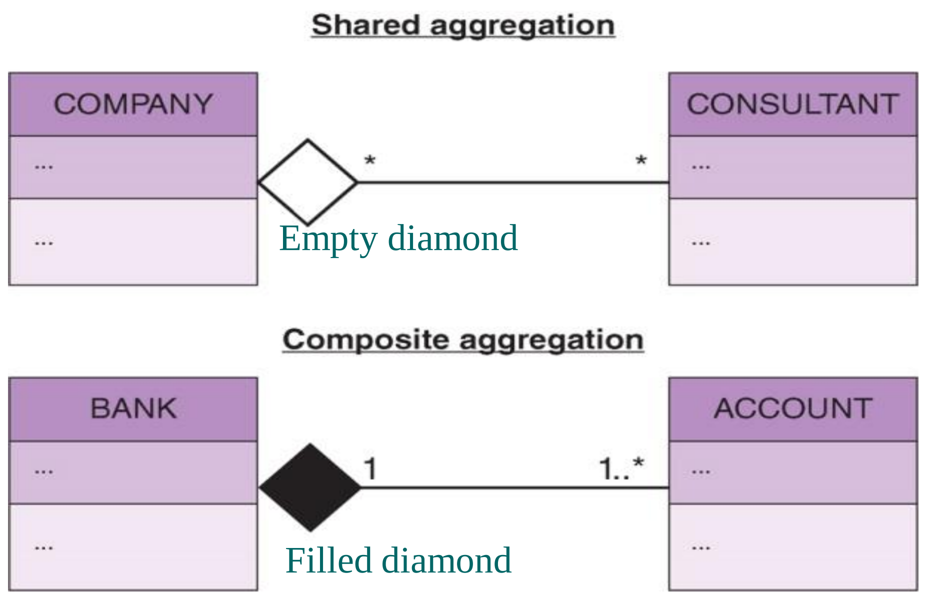
ARTIST
specialization characteristics: total/partial, disjoint/overlap
Specialization symbol
{partial; overlap}
SINGER
ACTOR

## UML Example with Specialization / Generalization



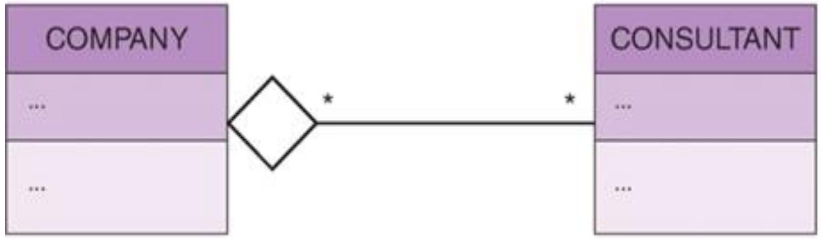
## Aggregation

- **Aggregation** represents a composite to part relationship whereby a composite class contains a part class
- Two types in UML:
  - **shared aggregation** (a.k.a. aggregation)
  - **composite aggregation** (a.k.a. composition)



## Shared Aggregation (a.k.a. Aggregation)

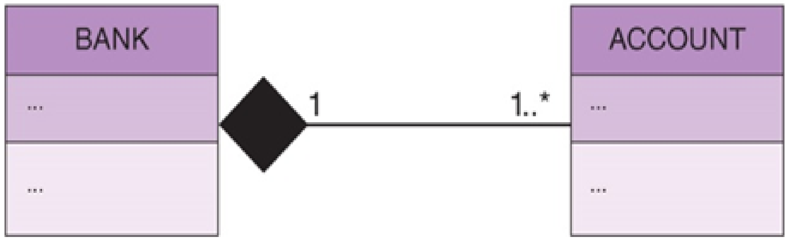
- In **shared aggregation**, the part object can simultaneously belong to multiple composite objects
- Maximum multiplicity at the composite side is undetermined
- The part object can also occur without belonging to a composite object
- **Loose coupling** between both classes
- Example



  - A consultant can work for multiple companies.
  - When a company is removed, any consultants that worked for it remain in the database

## Composite Aggregation (a.k.a. Composition)

- In **composite aggregation**, the part object can only belong to one composite
- The maximum multiplicity at the composite side is 1
  - e.g., engine and boat,  engine and car
- The minimum multiplicity can be either 1 or 0
- **Tight coupling** between both classes. The part object will be automatically removed when the composite object is removed.
- Example



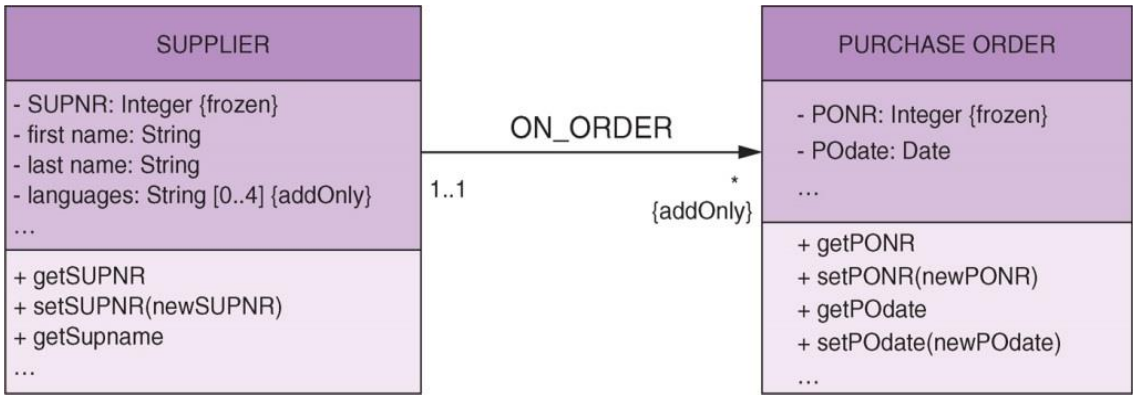
  - An account is tightly coupled to one bank only.
  - When the bank is removed, all connected account objects disappear as well.

## Outline

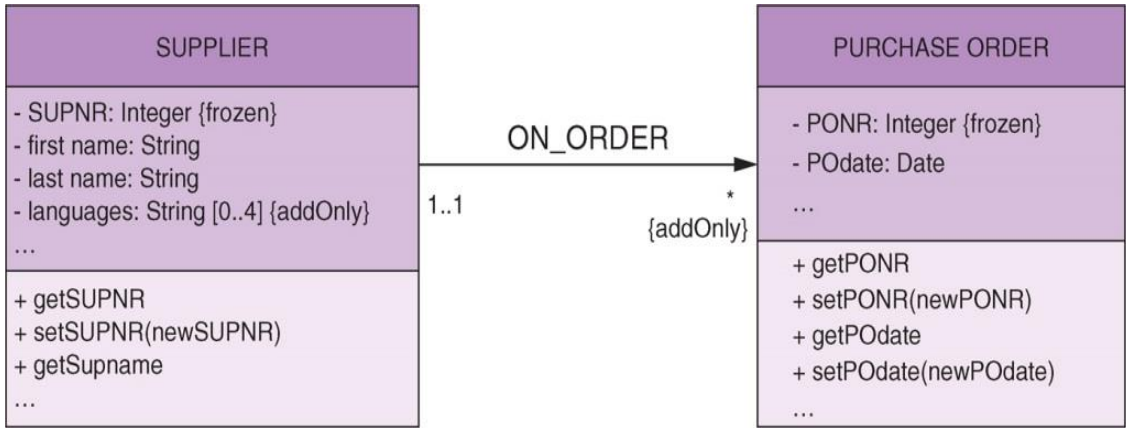
- Phases of Database Design
- Entity Relationship (ER) Model
- Enhanced Entity Relationship (EER) Model
- UML Class Diagram
  - Origin, Recap of Object Orientation
  - Classes, Variables, Access Modifiers
  - Associations
  - Specialization / Generalization
  - Aggregation
  - ☞ **Advanced UML Modeling Concepts**
  - UML versus EER
- Case Studies

## Changeability property

- The **changeability property** specifies the type of operations which are allowed on either variable values or links
- Three common choices
  - **default** which allows any type of edit
  - **addOnly** which only allows additional values or links to be added (so no deletions)
  - **frozen** which allows no further changes once the value or link is established



## Example



- The frozen characteristics on the supplier and purchase order number specifies that once a value has been assigned to either of them, it can no longer change.
- The language variable of the SUPPLIER class defines a set of language supplier can understand. It is defined as addOnly since languages can only be added and not removed from it.
- The addOnly characteristic on the ON_ORDER association specifies that for a given supplier, purchase orders can only be added and not removed.

## Object Constraint Language (OCL)

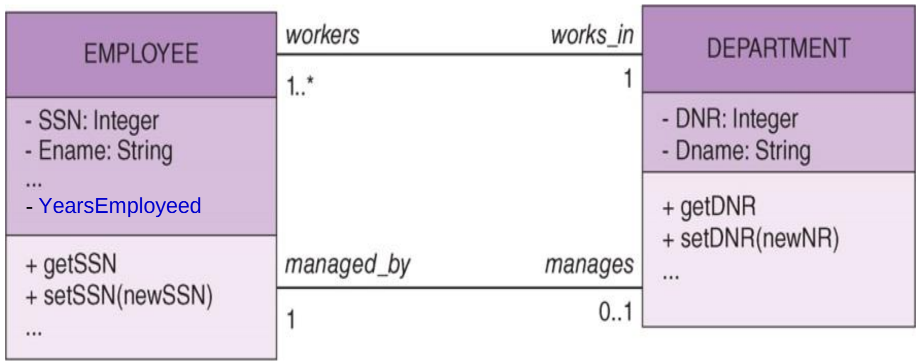
- The **Object Constraint Language (OCL)** can be used to specify various types of constraints
- The OCL constraints are defined in a declarative way
- The OCL constraints specify what must be true, but not how this should be accomplished.
  - no control flow or procedural code is provided
- Purpose: The OCL constraints can be used to specify invariants for classes, pre- and post-conditions for methods, to navigate between classes, or to define constraints on operations
- See http://www.omg.org/spec/OCL/

## OCL: Class Invariant, Pre/Post-Condition

- A **class invariant** is a constraint which holds for all objects of a class
  - E.g., SUPPLIER: SUPSTATUS>100
    - The supplier status of each supplier object should be great than 100.
- **Pre-** and **post-conditions on methods** must be true when a method either begins or ends
  - E.g., Before the method *withdrawal* can be executed, the balance must be positive. After it has been executed, the balance must still be positive.

# OCL: Complex Constraints

- Example:



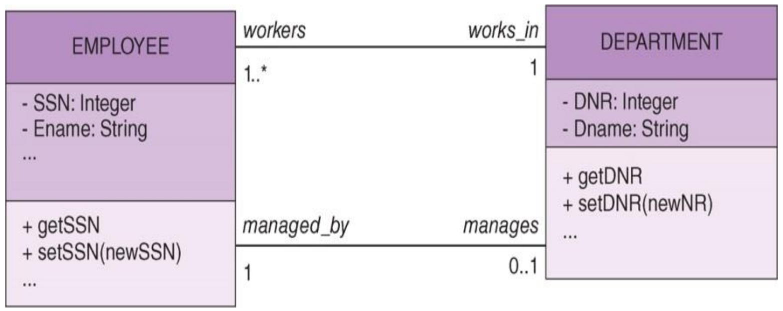
Two associations: Which employee works in which department; Which employee manages what department.

- **Constraint #1** : A manager of a department should be at least 10 years employed.

```
Context: Department
invariant: self.managed_by.yearsemployed>10
```

  - The context of this constraint is the DEPARTMENT class.
  - The keyword invariant
  - We use the keyword *self* to refer to an object of the DEPARTMENT class.
  - We then used the role name *managed_by* to navigate to the EMPLOYEE class and retrieve the *yearsemployeed* variable.

## Complex Constraints (Cont.)



- **Constraint #2**: A department should have at least 20 employees.

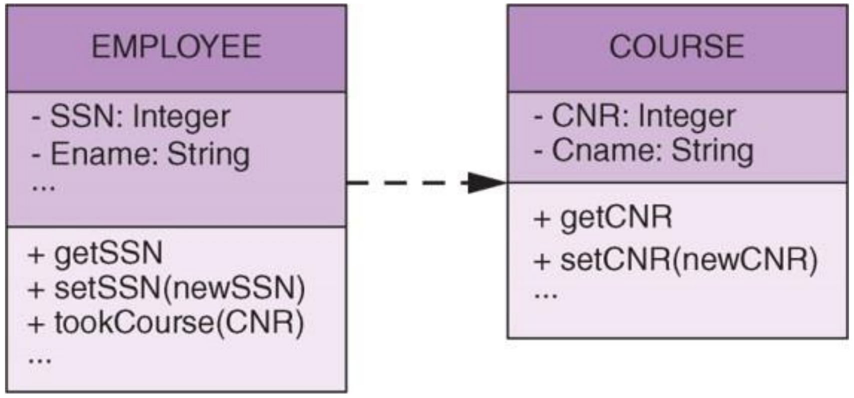
  ```
  Context: Department
  invariant: self.workers→size()>20
  ```

  - The context of this constraint is the DEPARTMENT class.
  - The *self.workers* returns the set of employees working in a specific department
  - The *size* method is then used to calculate the number of members in the set
- **Constraint #3**: A manager of a department must also work in the department

  ```
  Context: Department
  Invariant: self.managed_by.works_in=self
  ```

## Dependency Relationship

- In UML, **dependency** defines a "using" relationship which states that a change in the specification of a UML modeling concept may affect another modeling concept that uses it
- Dependency is denoted by a *dashed line* in the UML diagram.
- When an object of one class uses an object of another class in its methods, but the referred object is not stored in any variable, the dependency is used.



  - The EMPLOYEE class includes a method, tookCourse, that determine whether an employee took a particular course represented by the input variable CNR. An employee object makes use of a course object in the method. So there are dependency between both classes.

## Outline

- Phases of Database Design
- Entity Relationship (ER) Model
- Enhanced Entity Relationship (EER) Model
- UML Class Diagram
  - Origin, Recap of Object Orientation
  - Classes, Variables, Access Modifiers
  - Associations
  - Specialization / Generalization
  - Aggregation
  - Advanced UML Modeling Concepts
  - ☞ **UML versus EER**
- Case Studies

## UML Class Diagram vs. EER

| UML class diagram | EER model |
|---|---|
| Class | Entity type |
| Object | Entity |
| Variable | Attribute type |
| Variable value | Attribute |
| Method | - |
| Association | Relationship type |
| Link | Relationship |

## UML Class Diagram vs. EER

| **UML class diagram** | | **EER model** | |
|---|---|---|---|
| Qualified Association | | Weak entity type | |
| Specialization/Generalization | | Specialization/Generalization | |
| Aggregation | | Aggregation (Composite/Shared) | |
| OCL | | - | |
| Multiplicity | * | Cardinality | 0..N |
| | 0..1 | | 0..1 |
| | 1..* | | 1..N |
| | 1 | | 1..1 |

## Outline

- ☐ Phases of Database Design
- ☐ Entity Relationship (ER) Model
- ☐ Enhanced Entity Relationship (EER) Model
- ☐ UML Class Diagram
- ☞ **Case Studies**

## Case Study I: A COMPANY Database Requirement

*Suppose to create a database based on the following (simplified) requirements of a COMPANY Database:*

- The company has many departments. Each department has a department name,  a department number and an employee who manages the department. We keep track of the management start date of the manager. A department can be located in several locations.
- Each department controls a number of projects. Each project has a unique project name and a project number, and is located at a single location.

## COMPANY Database Requirement (Cont.)

- We store each employee’s name, social security number, address, salary, sex, and birthdate. Each employee works for one department but may work on several projects. We keep track of the number of hours per week that an employee currently works on each project. An employee can supervise other employees. We also keep track of the direct supervisor of each employee.
- Each employee may have a number of dependents. For each dependent, we keep track of their name, sex, birthdate, and relationship to the employee, e.g., father and mother.

## COMPANY ER Diagram

# COMPANY UML Class Diagram

## Case Study II: Sober Scenario Case – EER Modeling

- **Entity types and Attribute types**
  - The CAR entity type has been specialized into SOBER CAR and OTHER CAR.
  - Sober cars are owned by Sober, whereas other cars are owned by customers.
  - The RIDE entity type has been specialized into RIDE HAILING and RIDE SHARING.
  - The shared attribute types between both RIDE HAILING and RIDE SHARING subclasses are RIDE-NR (which is the key attribute type), PICKUP-DATE-TIME, DROPOFF-DATE-TIME, DURATION, PICKUP-LOC, DROPOFF-LOC, DISTANCE, and FEE.
    - DURATION is a derived attribute type since it can be derived from PICKUP-DATE-TIME and DROPOFF-DATE-TIME.
    - DISTANCE is not a derived attribute type since there could be multiple routes between a pick-up location and a drop-off location.

## Case Study II: EER Modeling (Cont.)

- **Entity types and Attribute types** (cont.)
  - Three attribute types are added to the RIDE HAILING entity type: PASSENGERS (the number of passengers), WAIT-TIME (the effective wait time), and REQUEST-TIME (Sober App request or hand wave).
  - The LEAD_CUSTOMER relationship type is a 1:N relationship type between CUSTOMER and RIDE HAILING
  - The BOOK relationship type is an N:M relationship type between CUSTOMER and RIDE SHARING.
  - A car (e.g., Sober or other car) can be involved in zero to N accidents, whereas an accident can have one to M cars (e.g., Sober or other car) involved.
  - The DAMAGE AMOUNT attribute type is connected to the relationship type because it is dependent upon the car and the accident.

## The EER Model

- Develop the EER diagram based on the description of the case study.

## Discuss: Shortcomings of the EER Model

- Since the EER model is a snapshot in time, it cannot model temporal constraints.
  - E.g.,  The pick-up-date-time should always precede the drop-off-date-time; a customer cannot book a ride-hailing and ride-sharing service that overlap in time.
- The EER model cannot guarantee the consistency across multiple relationship types.
  - An example of a business rule that cannot be enforced in the EER model is a customer cannot book a ride-hailing or ride-sharing service with his/her own car.
- The EER model does not support domains
  - E.g., We cannot specify that the attribute type PASSENGERS is an integer with a minimum value of 0 and a maximum value of 6.
  - Our EER model does not specify that the maximum number of passengers for a ride-share service is ten or that the WAIT-TIME is only relevant for Sober App requests and should be zero in case of hand-wave requests.

## Case Study II: UML Modeling

- **Classes, Variables, and Methods**
  - The class INVOLVED is an association class between CAR and ACCIDENT.
  - To enforce information hiding, the access modifiers of all variables have been set to private.
  - Getter and setter methods are used to access them.
  - In the RIDE class: CalcDuration  method, which calculates the derived variable duration.
  - In the RIDE-SHARING class: NumberOfCustomers method, which returns the number of customers for a ride-share service.
  - In the CUSTOMER class: Top5CustomersHail and Top5CustomersShare methods, which returns the top five customers for ride-hailing and ride-sharing services, respectively.

## Case Study II: UML Modeling

- **Classes, Variables, and Methods** (cont.)
  - In the CAR class: NumberOfRides method, which returns the number of rides a car has serviced.
  - In the SOBER CAR class: NumberOfSoberCars method, which returns the number of Sober cars in the database.
  - In the INVOLVED association class: GenerateReport method, which returns a report of which cars have been involved in what accident.
  - In the ACCIDENT class: Top3AccidentHotSpots and Top3AccidentPeakTimes methods, which return the top three most common accident locations and timings, respectively.
  - All associations have been defined as bidirectional, which implies that they can be navigated in both directions.
  - We set the changeability property of the number variables (e.g., RIDE-NR, CAR-NR, etc.) to frozen, which means that once a value has been assigned to any of them, it can no longer be changed.

## Case Study II: UML Modeling

- **Enrich our UML model by adding OCL constraints.**
  - We define a class invariant for the RIDE HAILING class, which specifies that the number of passengers for a ride-hail service should be less than six:
    - RIDE-HAILING: PASSENGERS $\leq$ 6
  - Remember that the maximum number of passengers for a ride-share service is ten. This can be defined using the following OCL constraint:5
    - Context: RIDE SHARING
    - invariant: self.BOOK→size() $\leq$ 10
  - The constraint applies to every ride-sharing object, hence the keyword invariant. Other OCL constraints can be added for further semantic refinement.
- The UML specification is semantically richer than its EER counterpart. As an example, both passenger constraints cannot be enforced in the EER model. The UML class diagram also specifies the domains (e.g., integer, string, etc.) for each of the variables and includes methods, both of which were not possible in the EER model.

## Case Study II: UML Modeling

- **Enrich our UML model by adding OCL constraints.**
  - We define a class invariant for the RIDE HAILING class, which specifies that the number of passengers for a ride-hail service should be less than six:

    RIDE-HAILING: PASSENGERS ≤ 6
  - Remember that the maximum number of passengers for a ride-share service is ten. This can be defined using the following OCL constraint:5

    Context: RIDE SHARING

    invariant: self.BOOK→size() ≤ 10
  - The constraint applies to every ride-sharing object, hence the keyword invariant. Other OCL constraints can be added for further semantic refinement.

## The UML Class Model

- Develop the EER diagram based on the description of the case study.

## Case Study II: EER vs UML

- The UML specification is semantically richer than its EER counterpart. As an example, both passenger constraints cannot be enforced in the EER model. The UML class diagram also specifies the domains (e.g., integer, string, etc.) for each of the variables and includes methods, both of which were not possible in the EER model.

## Summary

- Phases of Database Design
- Entity Relationship (ER) model
- Enhanced Entity Relationship (EER) model
- UML Class Diagram