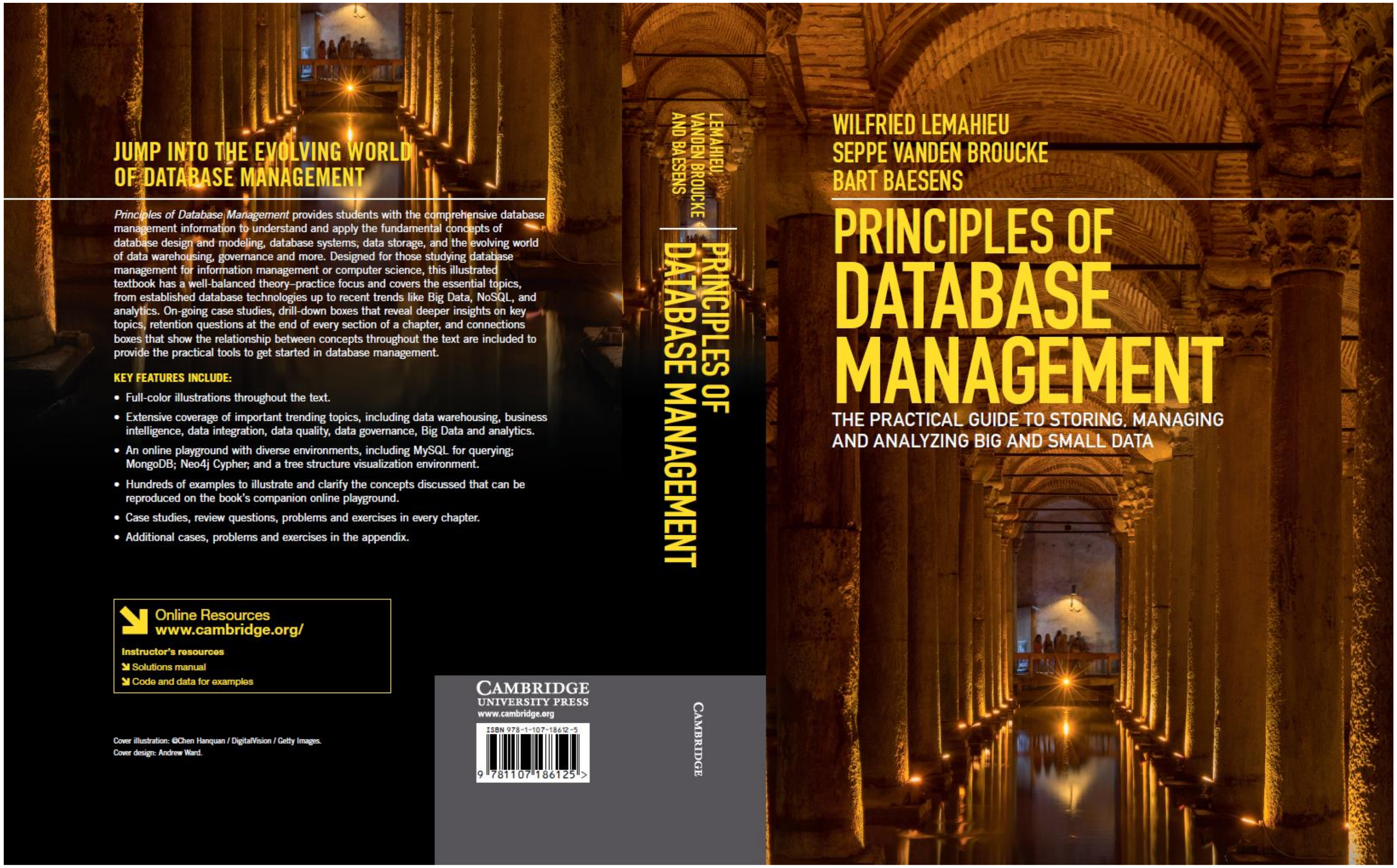
Conceptual Data Modeling using the (E)ER model and UML Class Diagram



JUMP INTO THE EVOLVING WORLD OF DATABASE MANAGEMENT

Principles of Database Management provides students with the comprehensive database management information to understand and apply the fundamental concepts of database design and modeling, database systems, data storage, and the evolving world of data warehousing, governance and more. Designed for those studying database management for information management or computer science, this illustrated textbook has a well-balanced theory–practice focus and covers the essential topics, from established database technologies up to recent trends like Big Data, NoSQL, and analytics. On-going case studies, drill-down boxes that reveal deeper insights on key topics, retention questions at the end of every section of a chapter, and connections boxes that show the relationship between concepts throughout the text are included to provide the practical tools to get started in database management.

KEY FEATURES INCLUDE:

- Full-color illustrations throughout the text.
- Extensive coverage of important trending topics, including data warehousing, business intelligence, data integration, data quality, data governance, Big Data and analytics.
- An online playground with diverse environments, including MySQL for querying; MongoDB; Neo4j Cypher; and a tree structure visualization environment.
- Hundreds of examples to illustrate and clarify the concepts discussed that can be reproduced on the book's companion online playground.
- Case studies, review questions, problems and exercises in every chapter.
- Additional cases, problems and exercises in the appendix.

Online Resources
www.cambridge.org/

Instructor's resources
- Solutions manual
- Code and data for examples

Cover illustration: ©Chen Hanquan / DigitalVision / Getty Images.
Cover design: Andrew Ward.

CAMBRIDGE UNIVERSITY PRESS
www.cambridge.org

ISBN 978-1-107-18612-5

LEMAHIEU VANDEN BROUCKE AND BAESENS PRINCIPLES OF DATABASE MANAGEMENT CAMBRIDGE

WILFRIED LEMAHIEU
SEPPE VANDEN BROUCKE
BART BAESENS

PRINCIPLES OF DATABASE MANAGEMENT

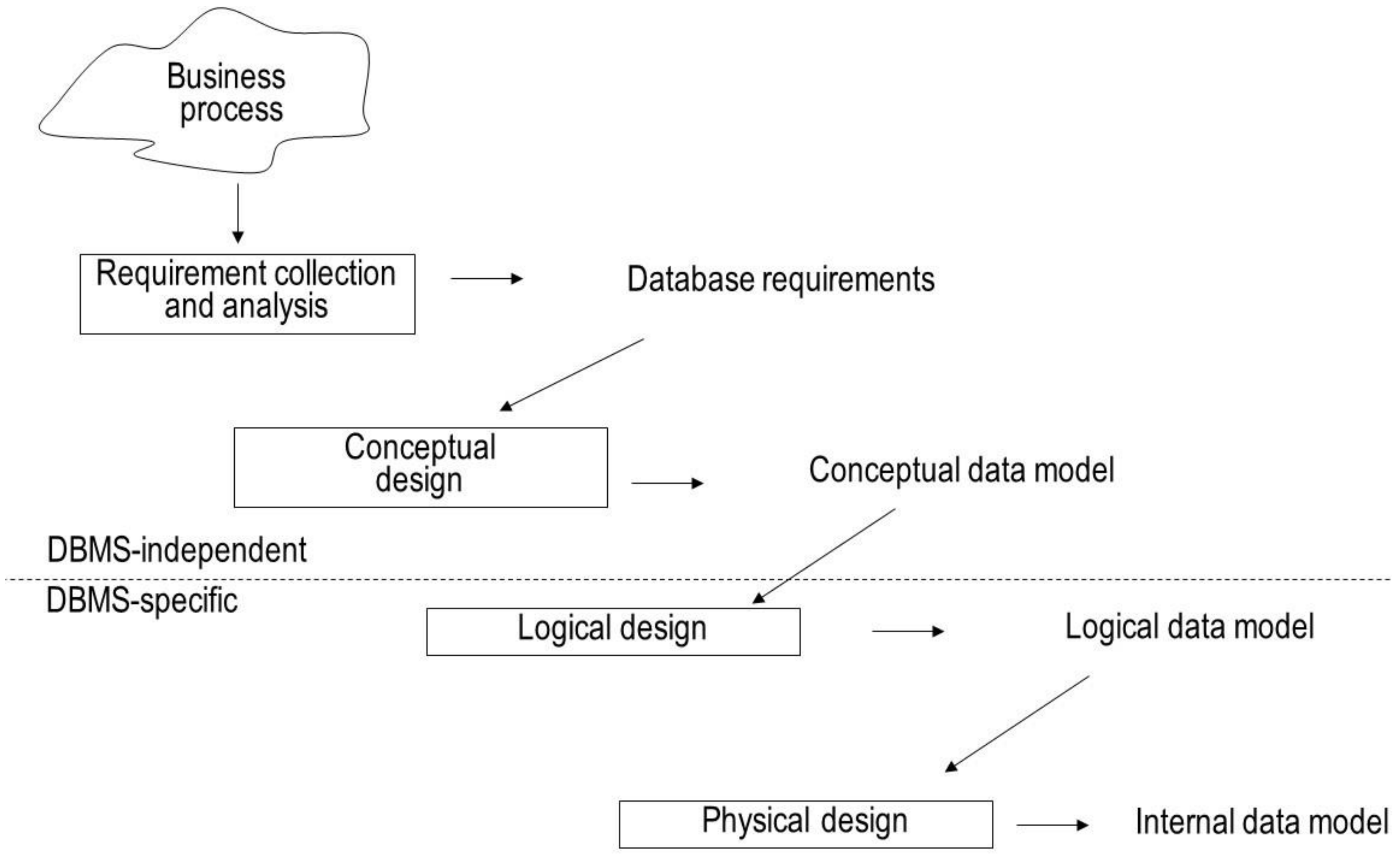
THE PRACTICAL GUIDE TO STORING, MANAGING AND ANALYZING BIG AND SMALL DATA

www.pdbmbook.com

Overview

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

Phases of Database Design

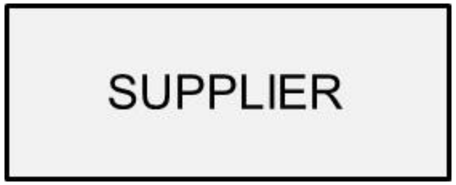


Entity Relationship (ER) model

- Entity Types
- Attribute Types
- Relationship Types
- Weak Entity Types
- Ternary Relationship Types
- Examples of the ER model
- Limitations of the ER model

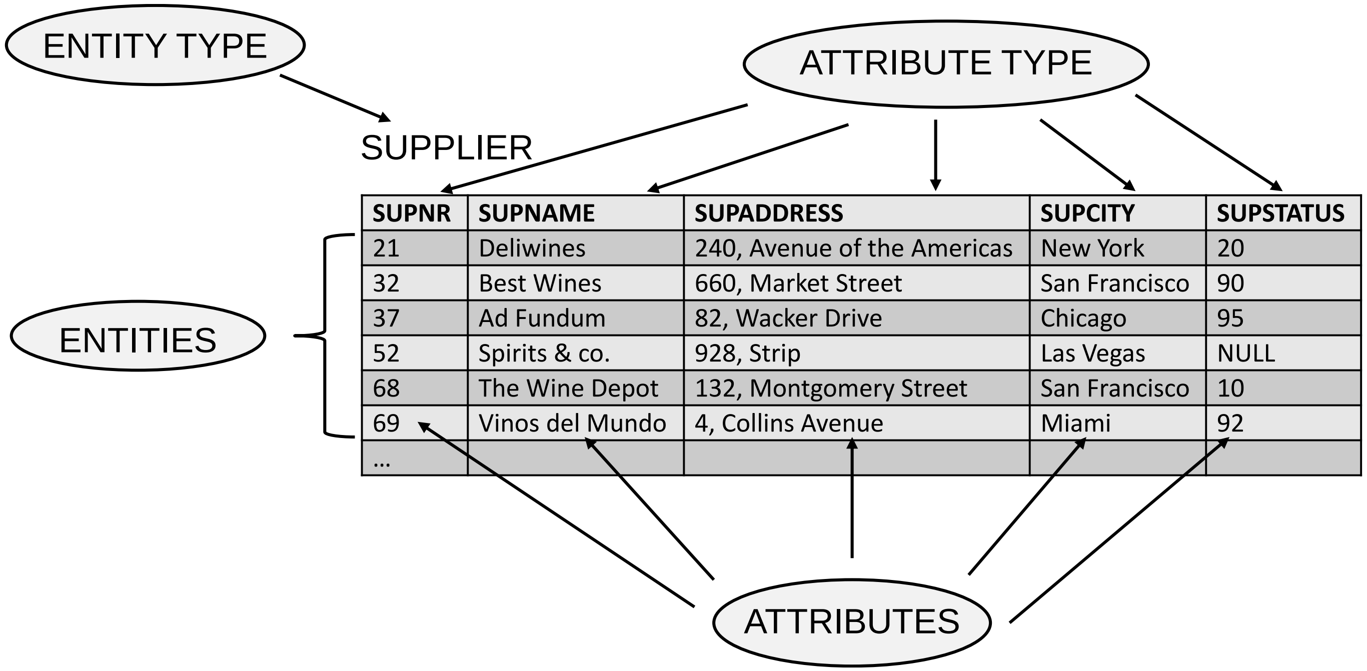
Entity Types

- Entity type represents a business concept with an unambiguous meaning to a particular set of users
  - Examples: supplier, student, product or employee
- Entity is one particular occurrence or instance of an entity type
  - Examples: Deliwines, Best Wines and Ad Fundum are entities from the entity type supplier



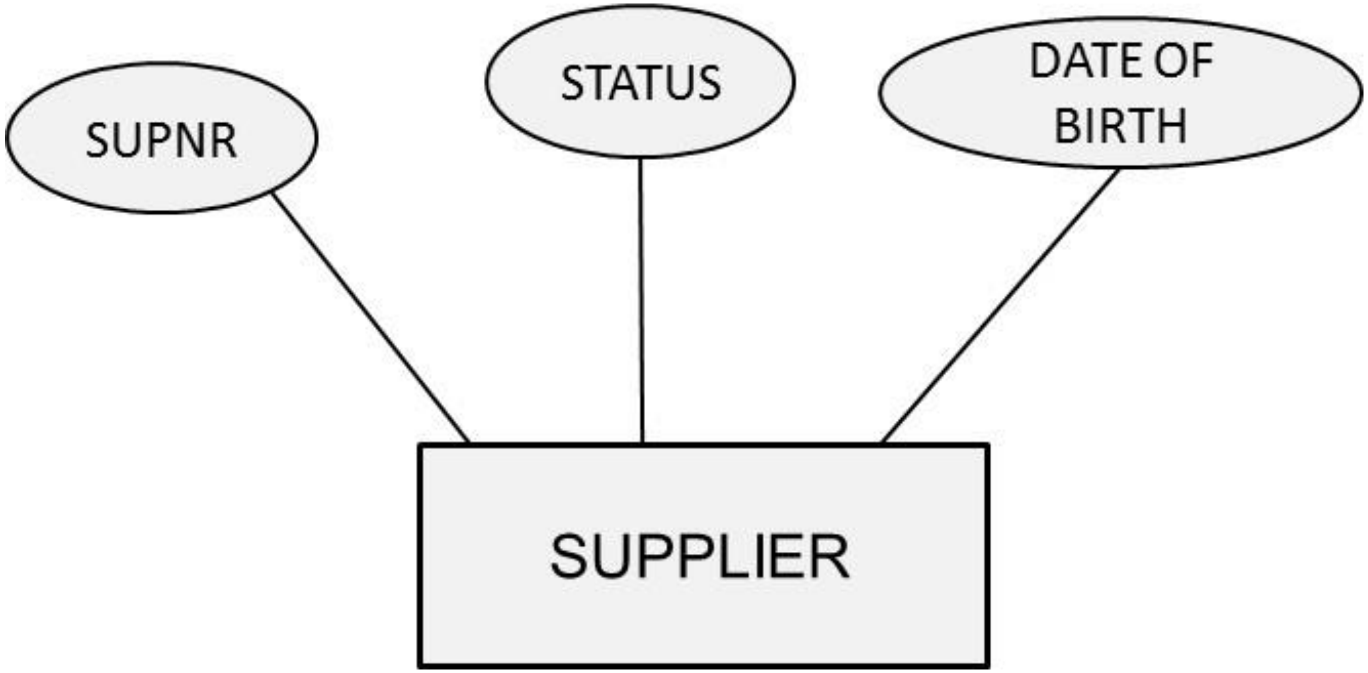
Attribute Types

- Attribute type represents a property of an entity type.
  - Example: name and address are attribute types of the entity type supplier
- Attribute is an instance of an attribute type



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



Attribute Types

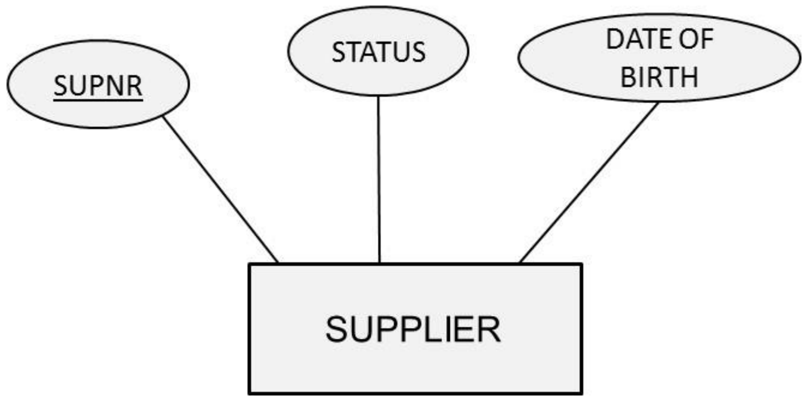
- Domains
- Key Attribute Types
- Simple versus Composite Attribute Types
- Single-Valued versus Multi-Valued Attribute Types
- Derived Attribute Type

Domains

- A domain specifies the set of values that may be assigned to an attribute for each individual entity
  - Example: gender: male and female
- A domain can also contain null values
  - null value: value is not known, not applicable or not relevant
- Domains are not displayed in an ER model

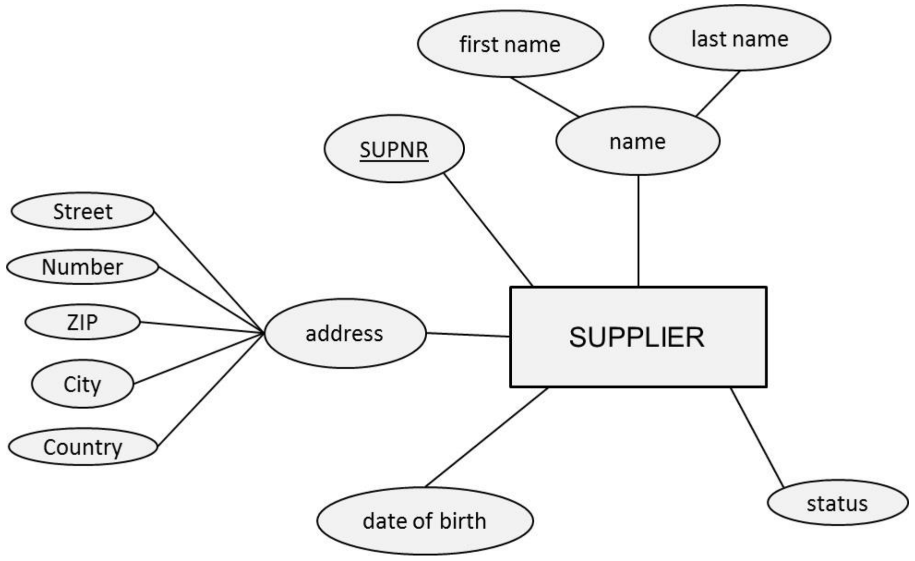
Key Attribute Types

- A key attribute type is an attribute type whose values are distinct for each individual entity
  - Examples: supplier number, product number, social security number
- A key attribute type can also be a combination of attribute types
  - Example: combination of flight number and departure date



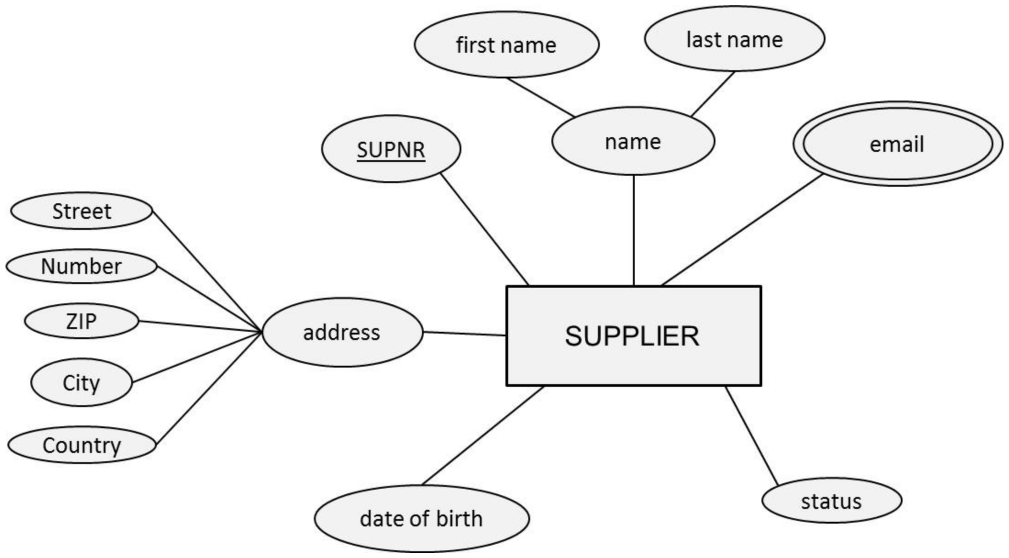
Simple versus Composite Attribute Types

- A simple or atomic attribute type cannot be further divided into parts
  - Examples: supplier number, supplier status
- A composite attribute type is an attribute type that can be decomposed into other meaningful attribute types
  - Examples: address, name



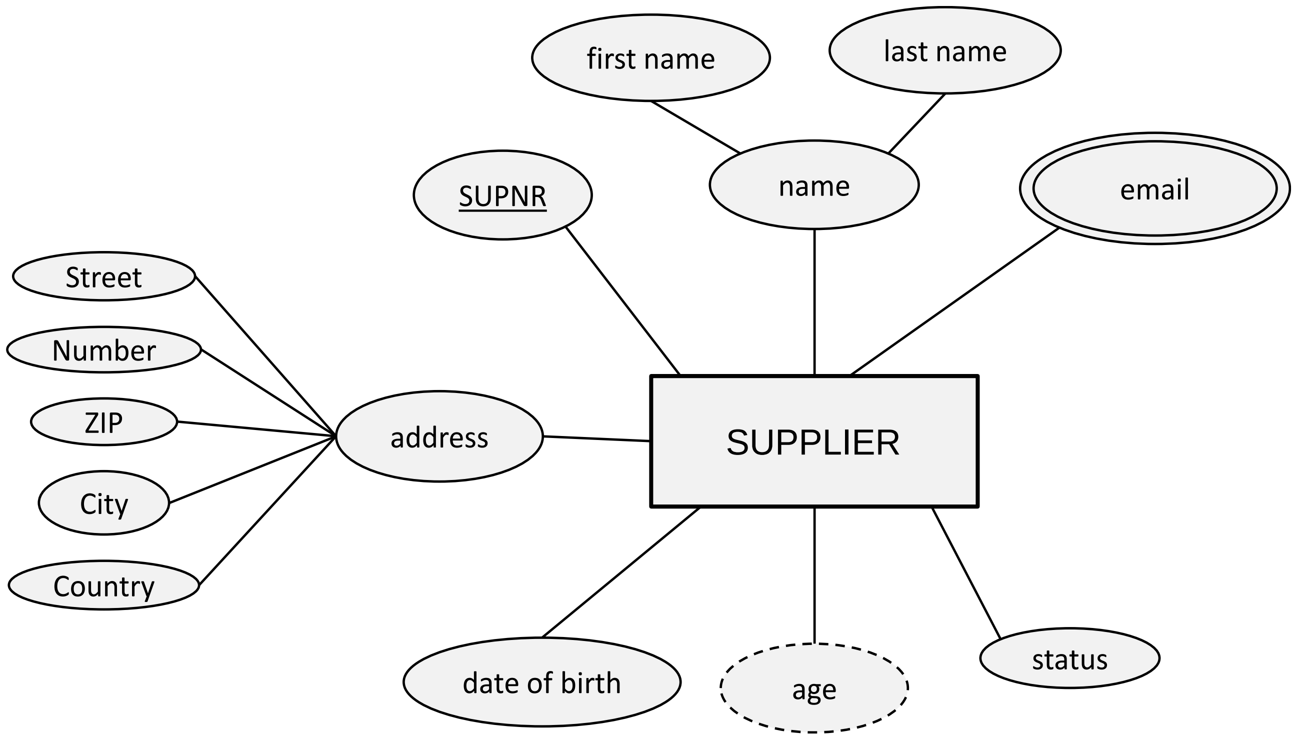
Single-Valued versus Multi-Valued Attribute Types

- A single-valued attribute type has only one value for a particular entity
  - Examples: product number, product name
- A multi-valued attribute type is an attribute type that can have multiple values
  - Example: email address



Derived Attribute Type

- A derived attribute type is an attribute type which can be derived from another attribute type
  - Example: age

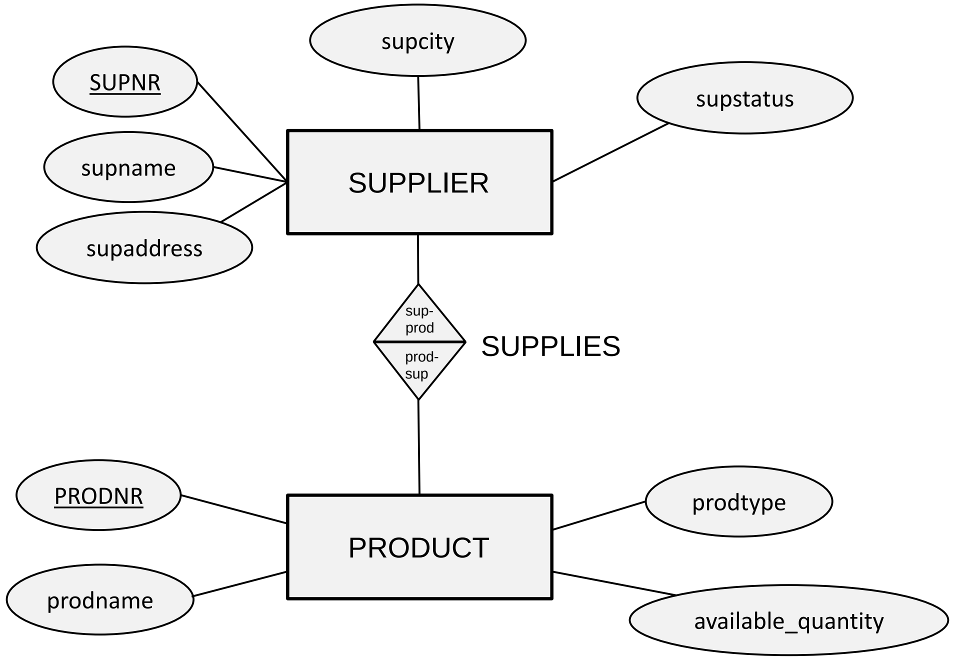


Relationship Types

- Definition
- Degree and Roles
- Cardinalities
- Relationship Attribute Types

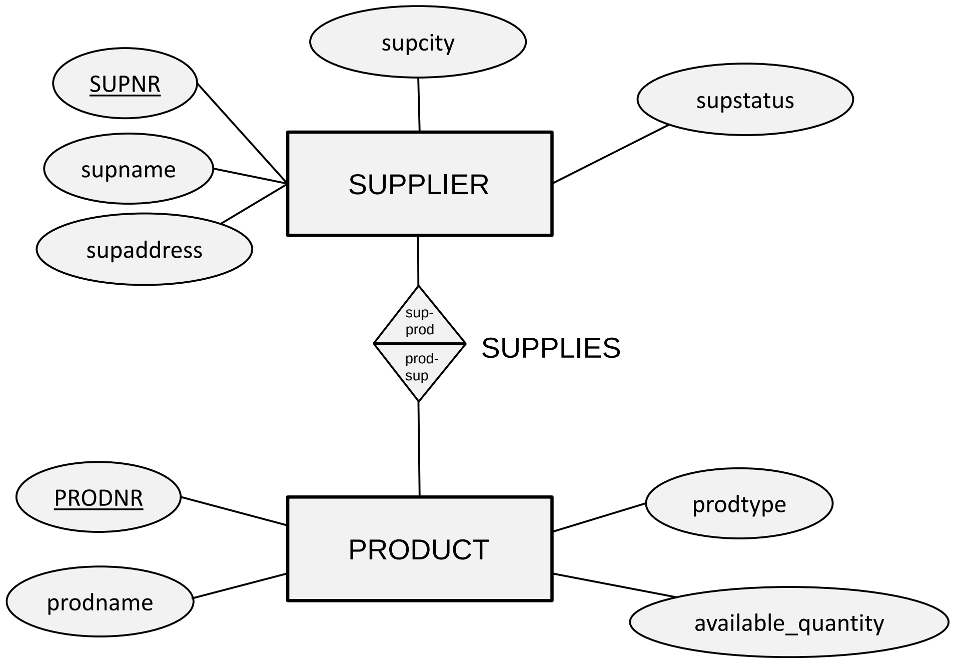
Definition

- 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

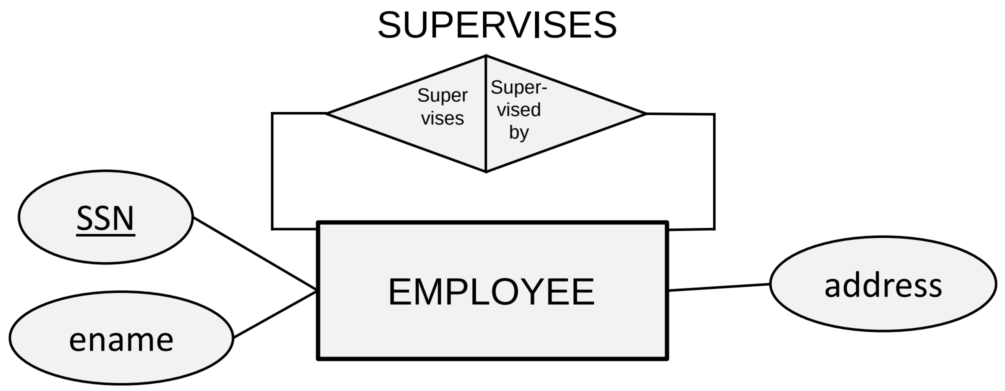


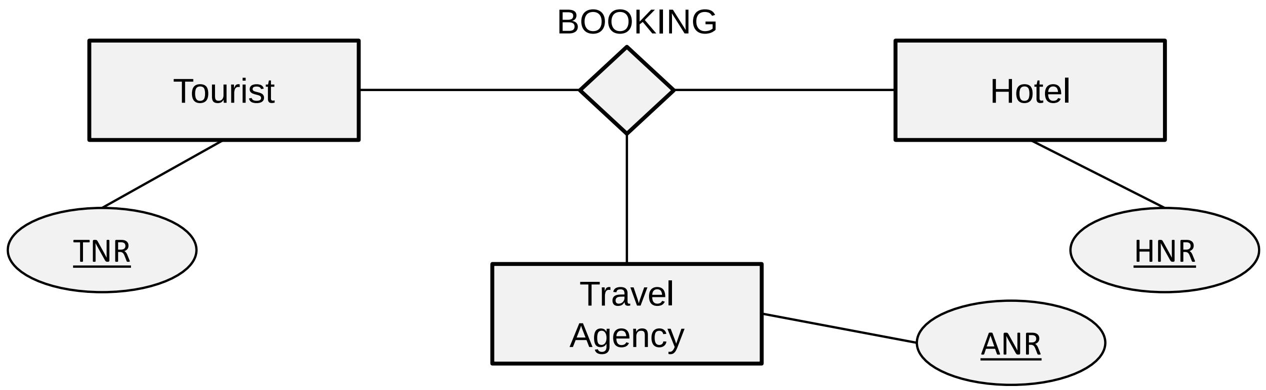
Degree and Roles

- The degree of a relationship type corresponds to the number of entity types participating in the relationship type
  - Unary: degree 1, binary: degree 2, ternary: degree 3
- The roles of a relationship type indicate the various directions that can be used to interpret it



Degree and Roles

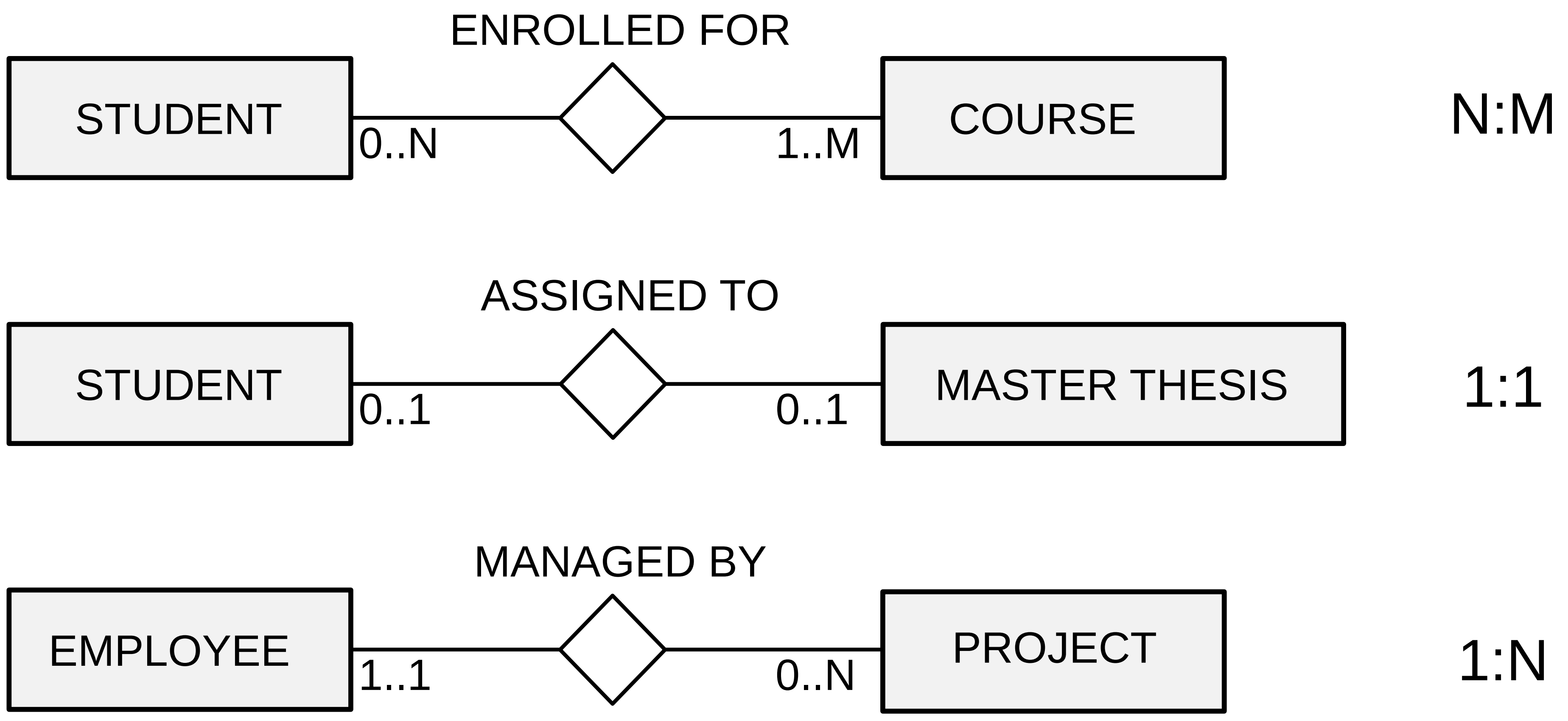




Cardinalities

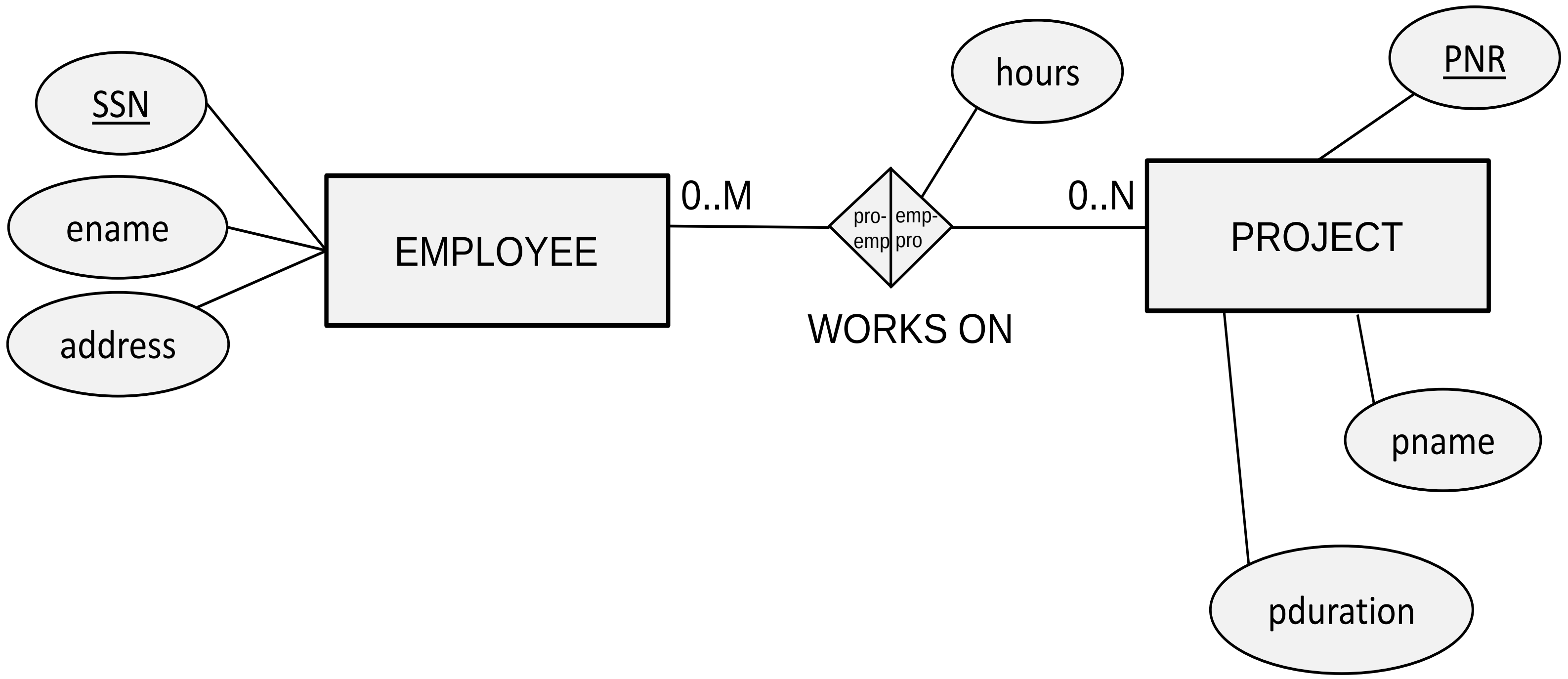
- Every relationship type can be characterized in terms of its cardinalities, which specify the minimum or maximum number of relationship instances that an individual entity can participate in
- Minimum cardinality can be 0 or 1
  - If 0: partial participation
  - If 1: total participation or existence dependency
- Maximum cardinality can be 1 or N
- Relationship types are often characterized by their maximum cardinalities
  - 4 options for binary relationship types: 1:1, 1:N, N:1 and M:N.

Cardinalities



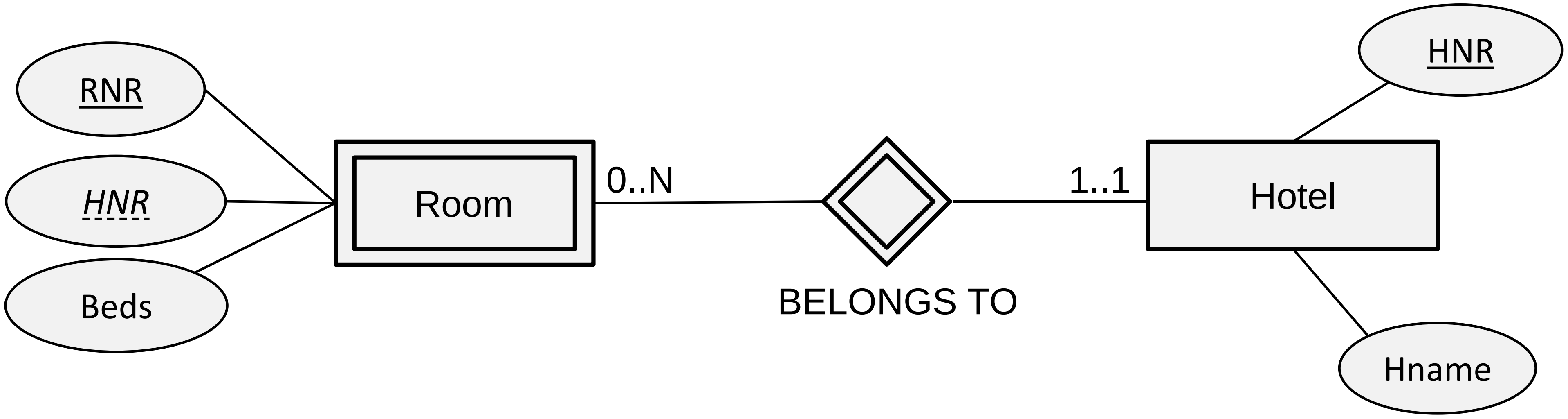
Relationship Attribute Types

- Relationship type can also have attribute types
- These attribute types can be migrated to one of the participating entity types in case of a 1:1 or 1:N relationship type



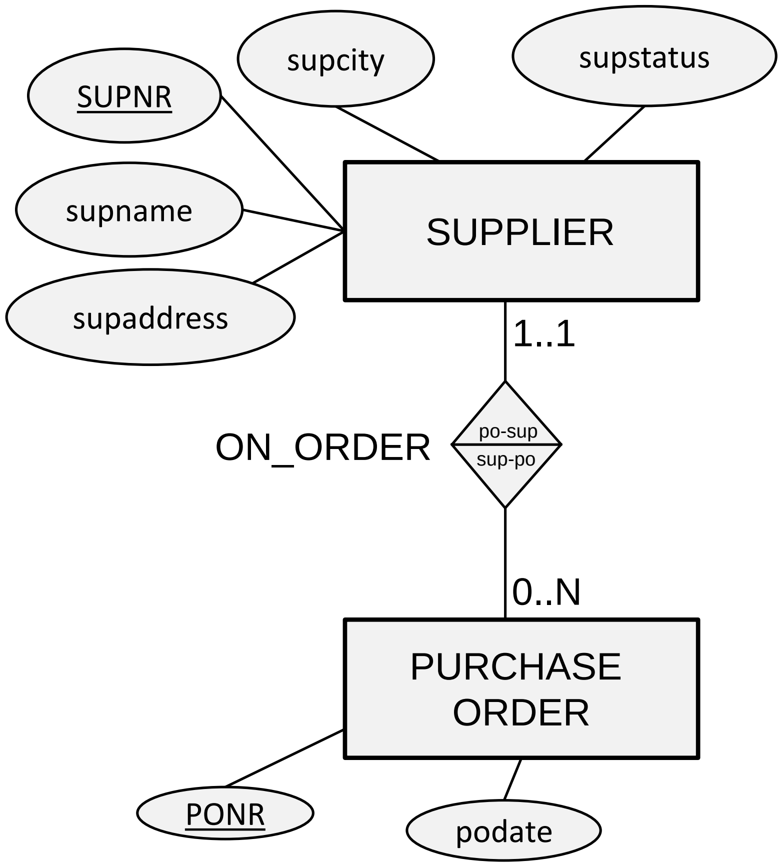
Weak Entity Type

- A strong entity type is an entity type that has a key attribute type
- A weak entity type is an entity type that does not have a key attribute type of its own
  - related to owner entity type from which it borrows an attribute type to make up a key attribute type



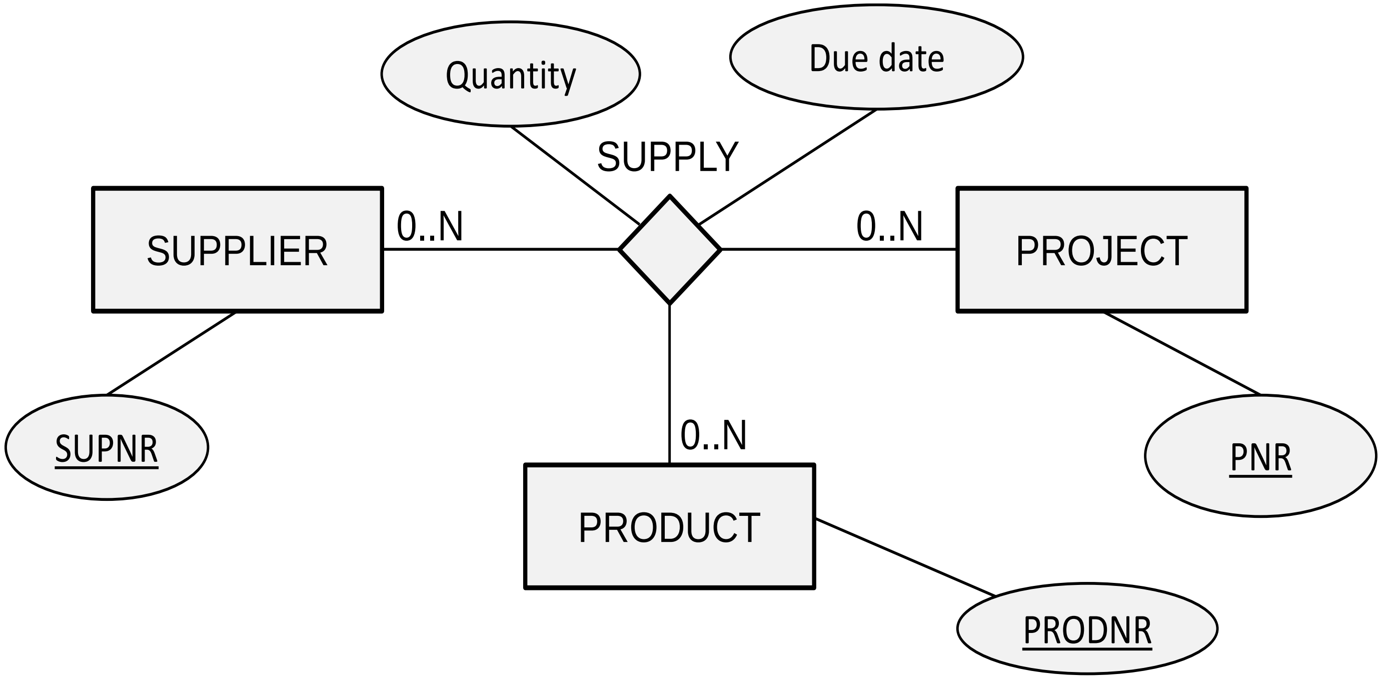
Weak Entity Type

- Weak entity type is always existent dependent from owner entity type (not vice versa!)

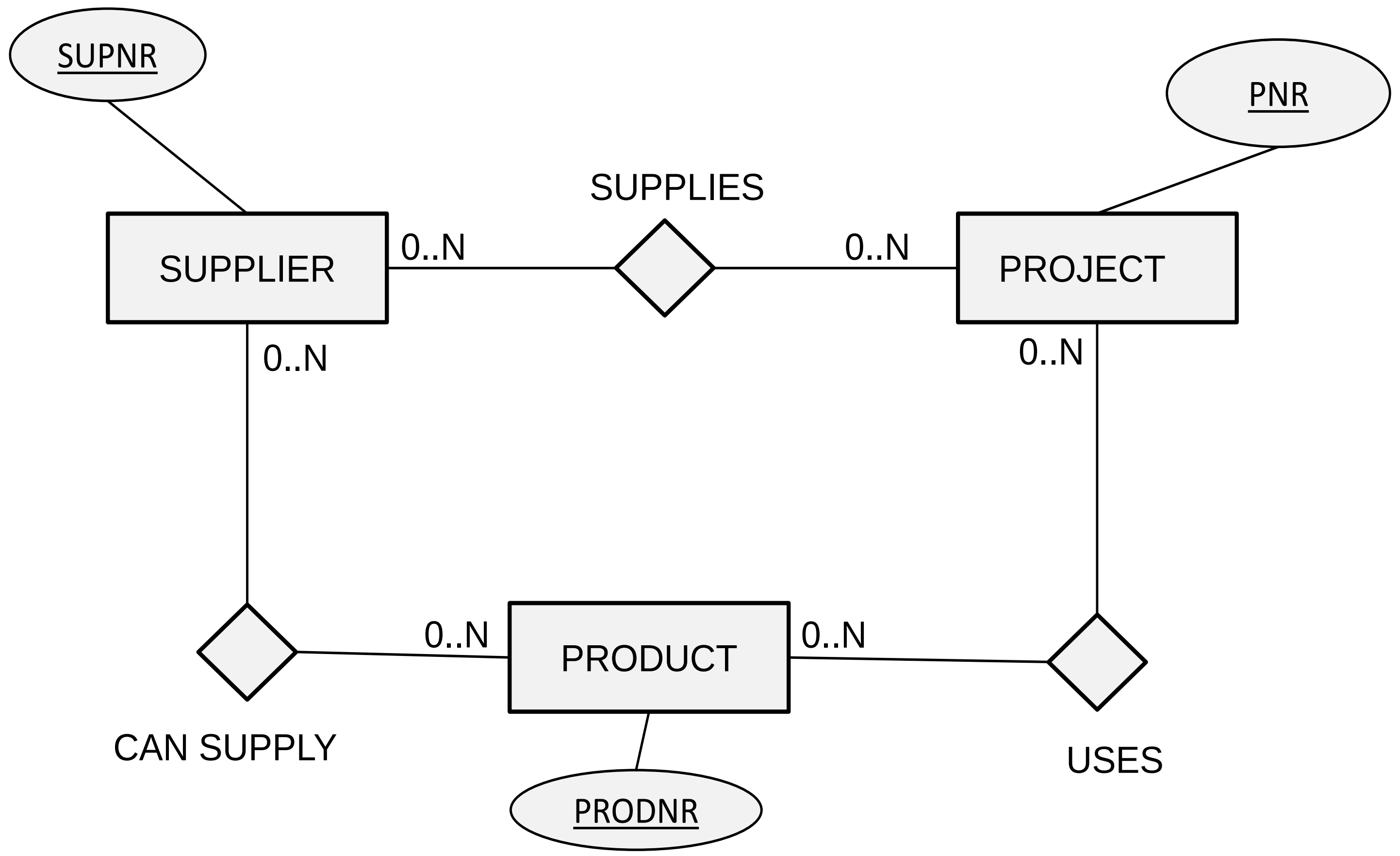


Ternary Relationship Types

- 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 supply 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.



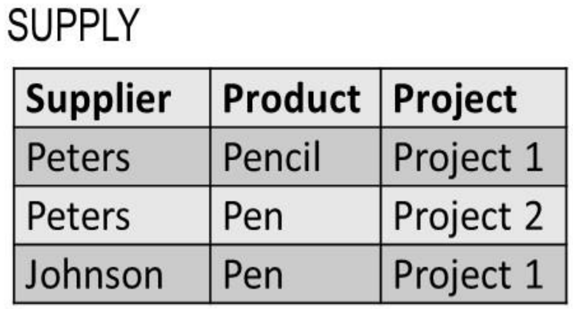
Ternary Relationship Types



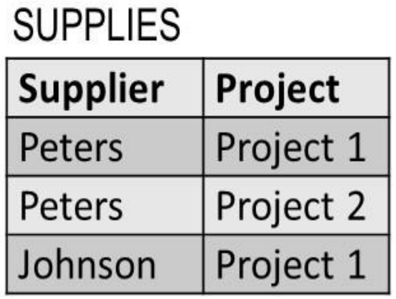
Note: loss of semantics!

Ternary Relationship Types

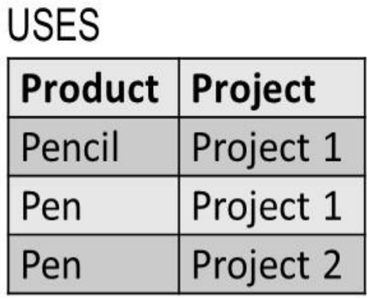
- Say we have two projects: project 1 uses a pencil and a pen, and project 2 uses a pen. Supplier Peters supplies the pencil for project 1 and the pen for project 2 whereas supplier Johnson supplies the pen for project 1.


SUPPLY

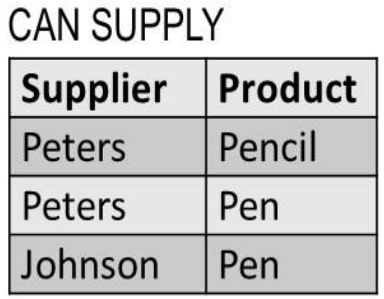
Supplier	Product	Project
Peters	Pencil	Project 1
Peters	Pen	Project 2
Johnson	Pen	Project 1


SUPPLIES

Supplier	Project
Peters	Project 1
Peters	Project 2
Johnson	Project 1


USES

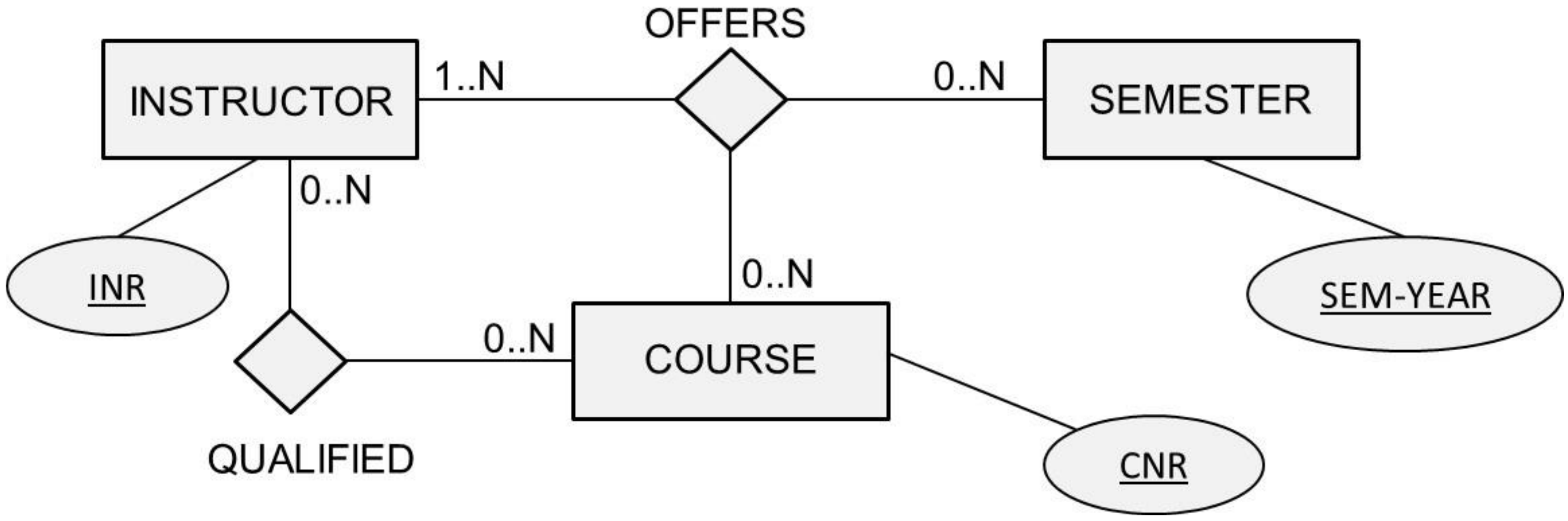
Product	Project
Pencil	Project 1
Pen	Project 1
Pen	Project 2


CAN SUPPLY

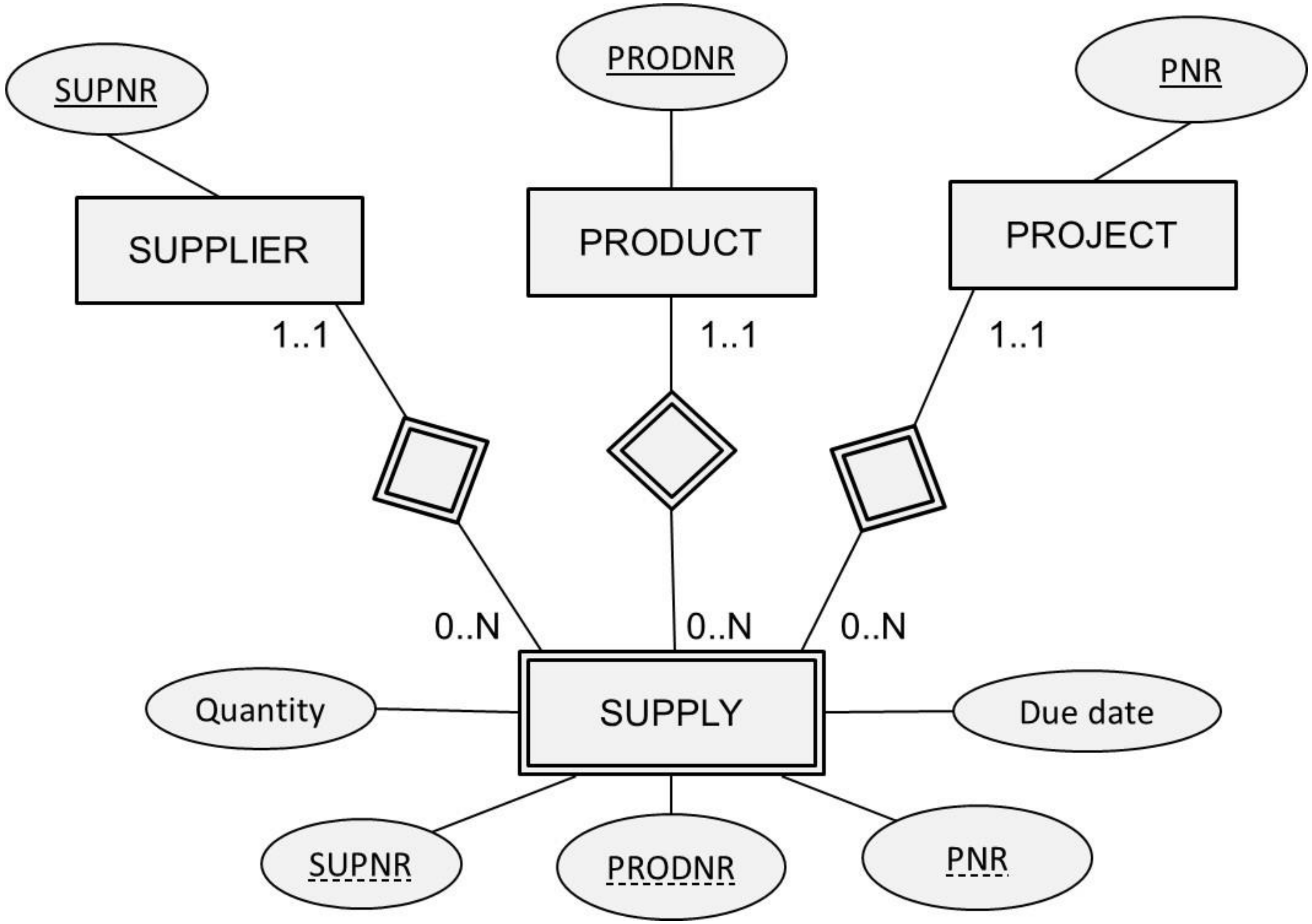
Supplier	Product
Peters	Pencil
Peters	Pen
Johnson	Pen

- From the binary relationship types, it is not clear who supplies the pen for project 1!

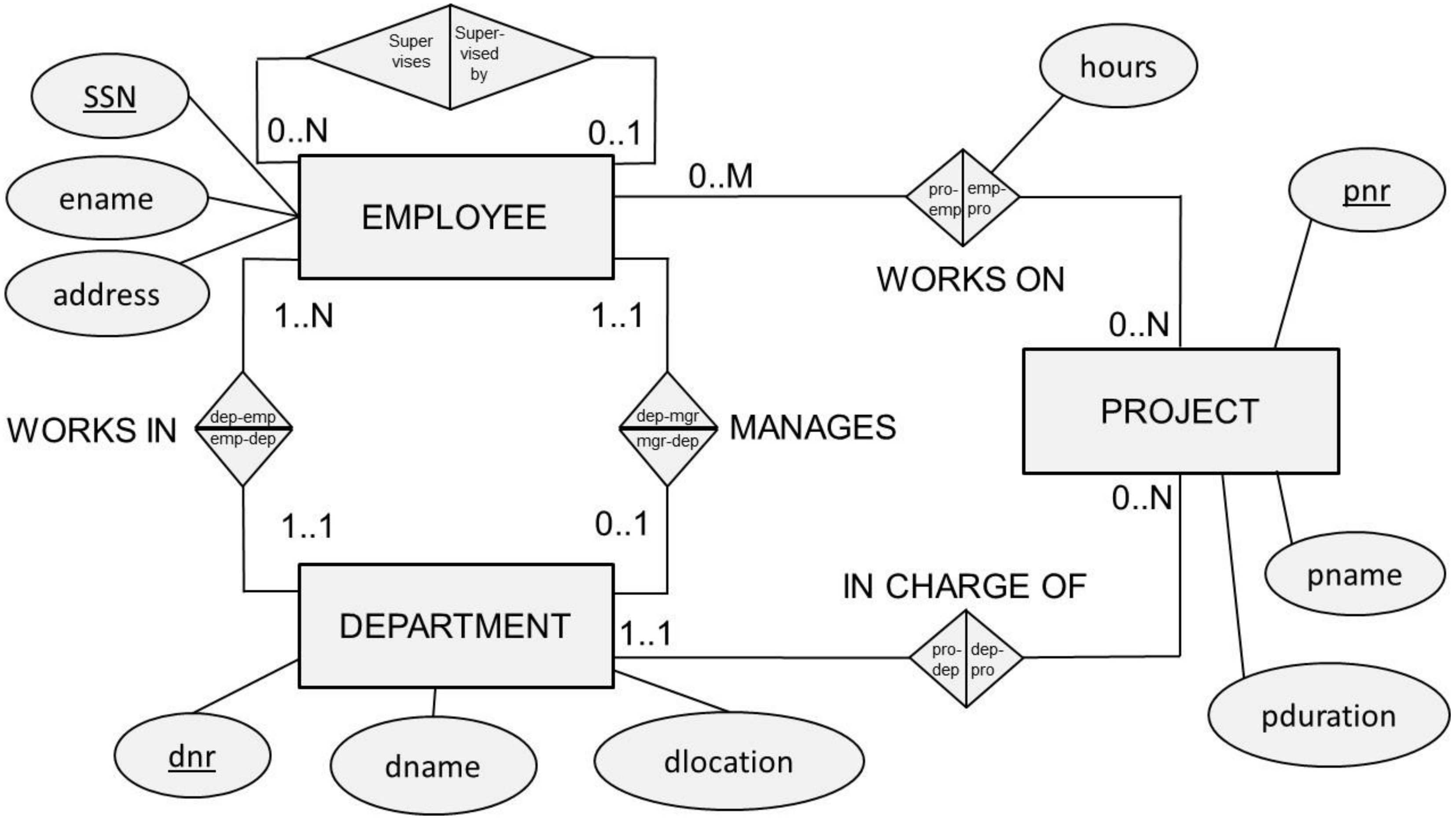
Ternary Relationship Types



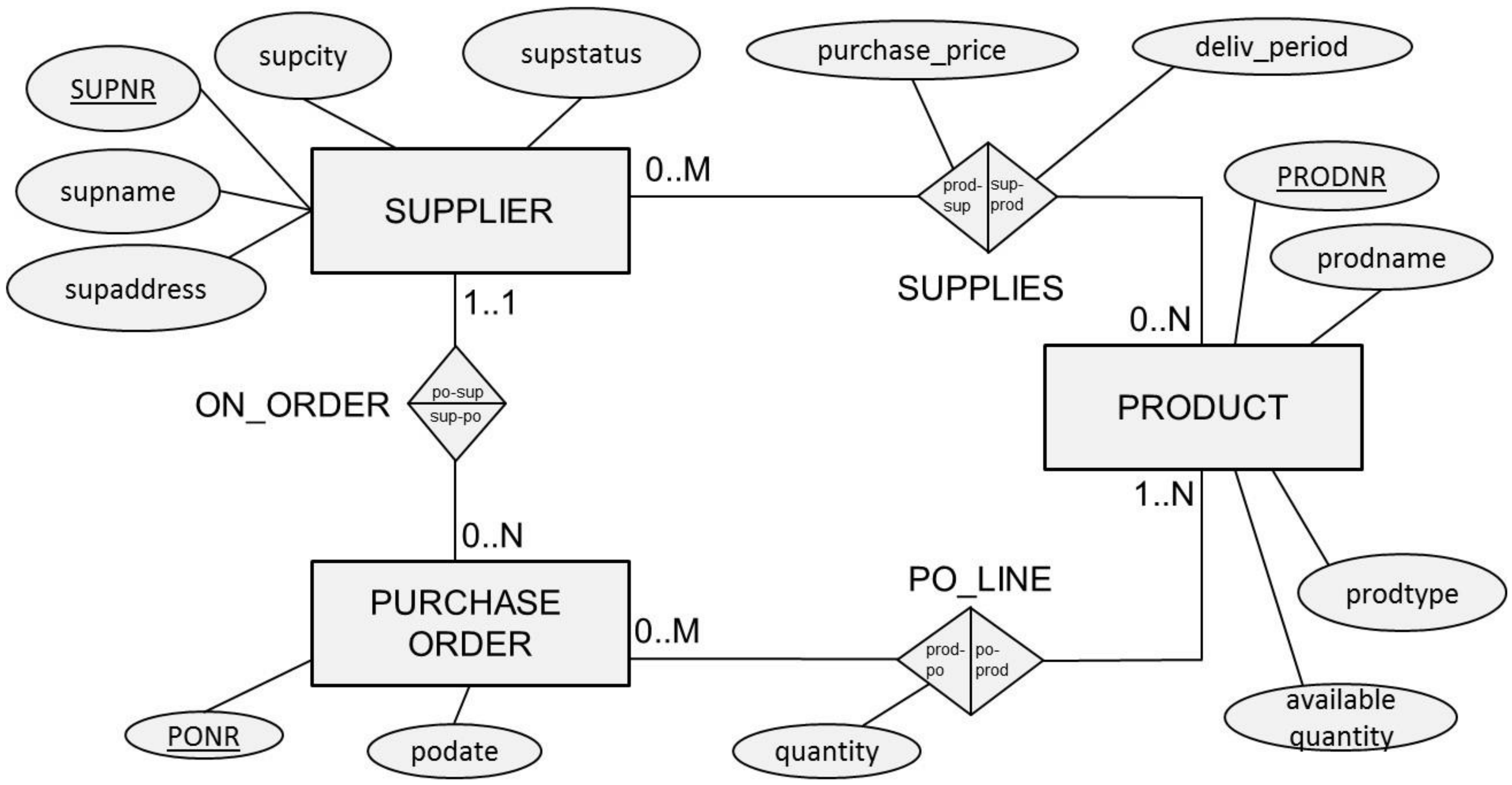
Ternary Relationship Types



Examples of the ER Model



Examples of the ER Model



Limitations of the ER model

- ER model presents a temporary snapshot and cannot model temporal constraints
  - Examples: a project needs to be assigned to a department after one month, an employee cannot return to a department of which he previously was a manager, a purchase order must be assigned to a supplier after two weeks, etc.
- ER model cannot guarantee the consistency across multiple relationship types
  - Examples: an employee should work in the department that he/she manages, employees should work on projects assigned to departments to which the employees belong, suppliers can only be assigned to purchase orders for products they can supply

Limitations of the ER model

- 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

Enhanced Entity Relationship (EER) Model

- Specialization/Generalization
- Categorization
- Aggregation
- Examples of the EER Model
- Designing the EER Model

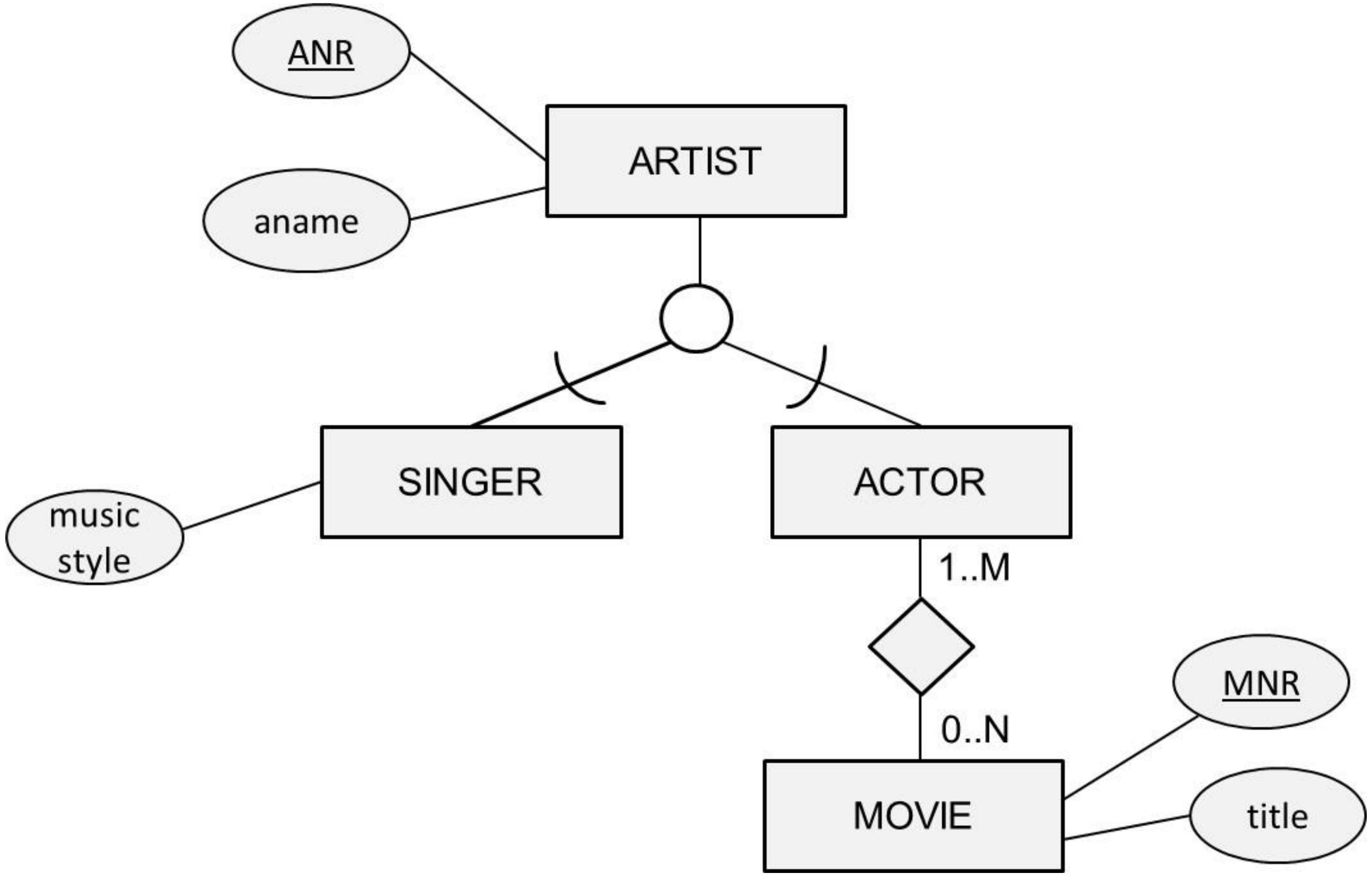
Specialization/Generalization

- Specialization refers to the process of defining a set of subclasses of an entity type
  - Example: ARTIST superclass with subclasses SINGER and ACTOR
- The specialization process defines an “IS A” relationship
- The specialization can then establish additional specific attribute types for each subclass
  - Example: singer can have a music style attribute type
- The specialization can also establish additional specific relationship types for each subclass
  - Examples: actor can act in movies, singer can be part of a band
- A subclass inherits all attribute types and relationship types from its superclass

Specialization/Generalization

- Generalization, also called abstraction, is the reverse process of specialization
  - Specialization corresponds to a top-down process of conceptual refinement
  - Generalization corresponds to a bottom-up process of conceptual synthesis

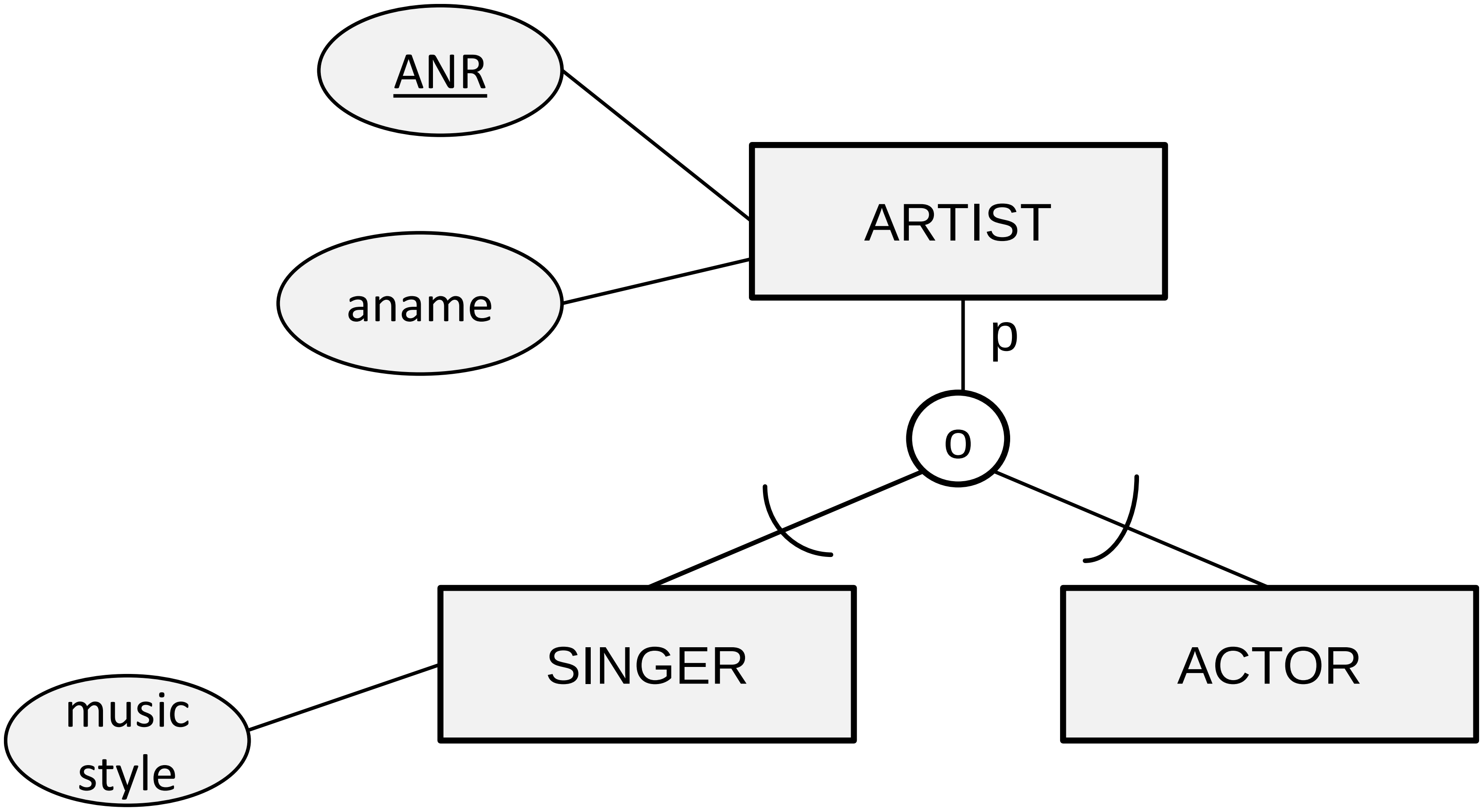
Specialization/Generalization



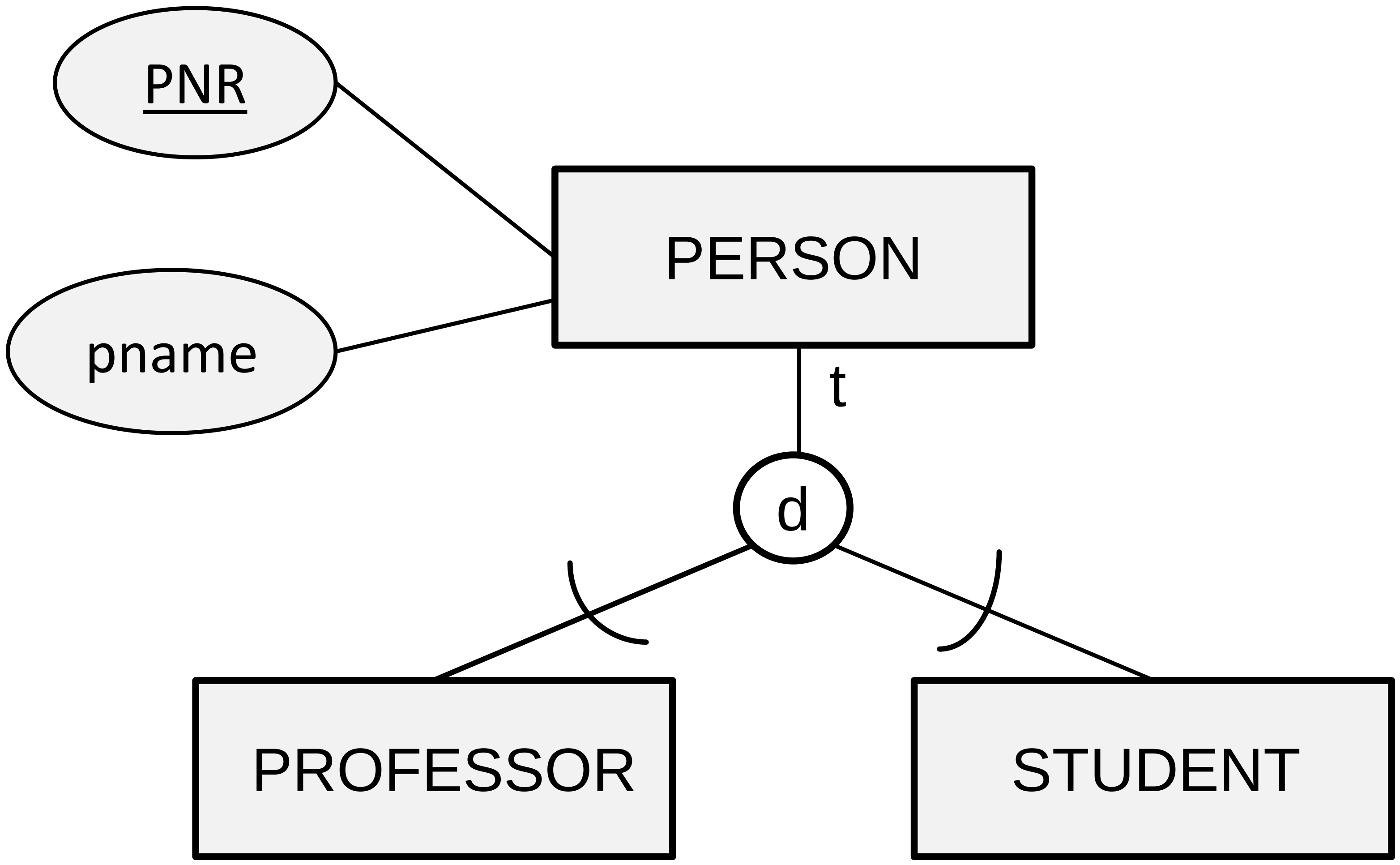
Specialization/Generalization

- The disjointness constraint specifies to what subclasses an entity of the superclass can belong to
  - A disjoint specialization is a specialization whereby an entity can be a member of at most one of the subclasses
  - An overlap specialization 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
  - A total specialization is a specialization whereby every entity in the superclass must be a member of some subclass
  - A partial specialization allows an entity to only belong to the superclass and to none of the subclasses

Specialization/Generalization

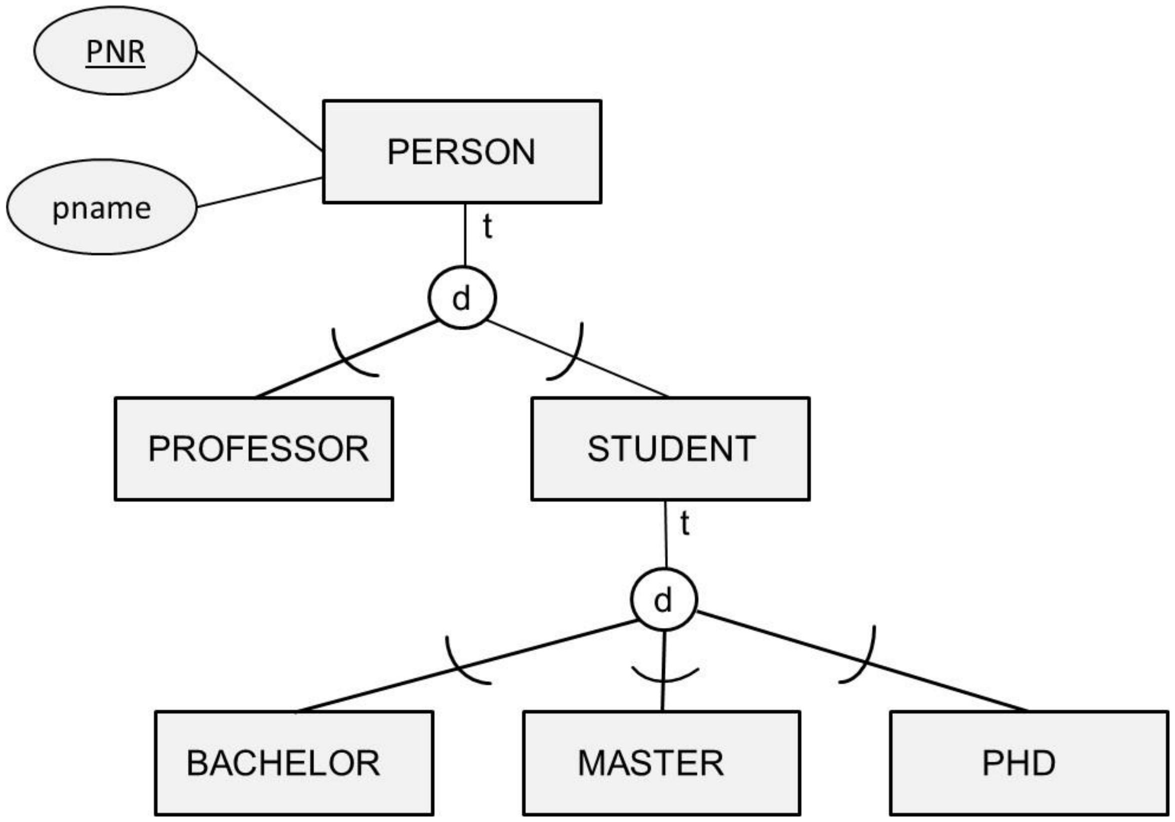


Specialization/Generalization



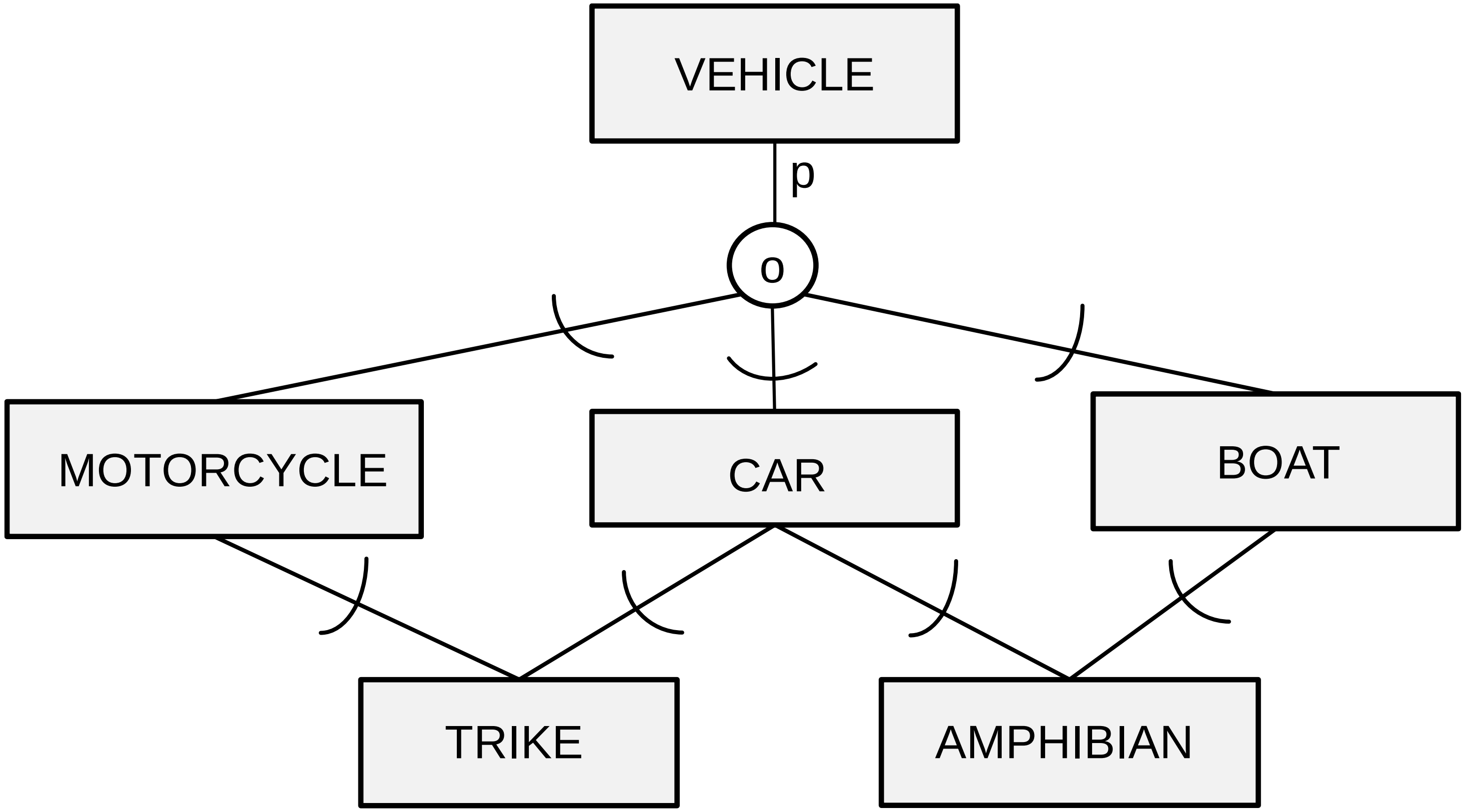
Specialization/Generalization

- 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



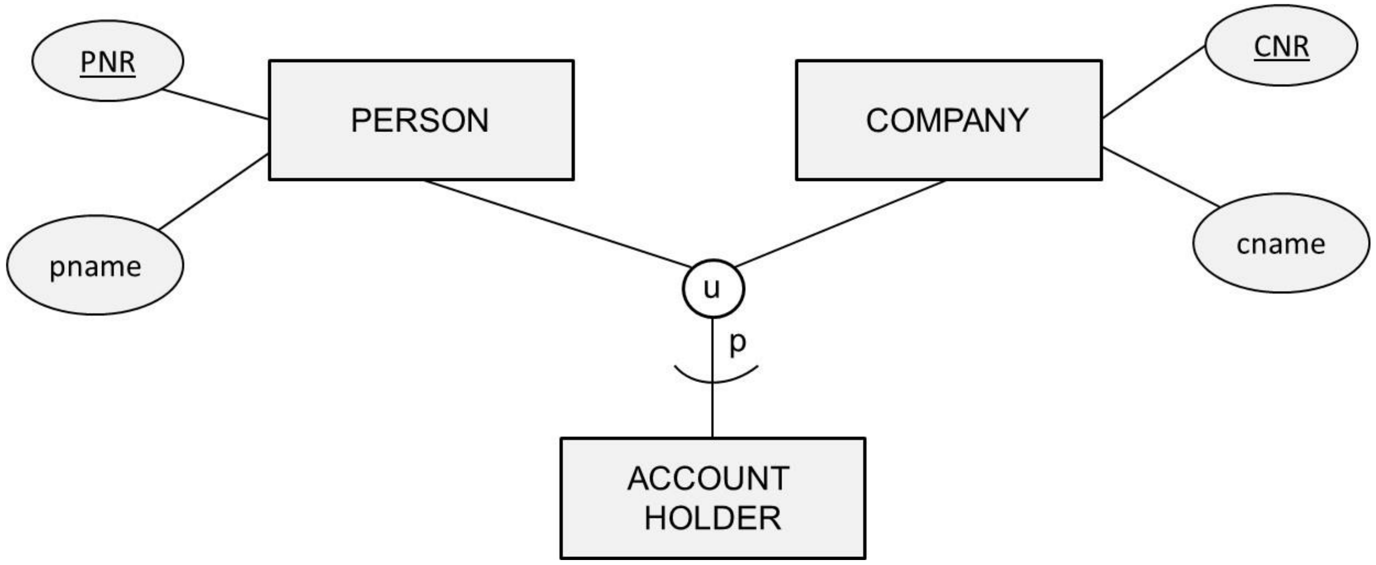
Specialization/Generalization

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



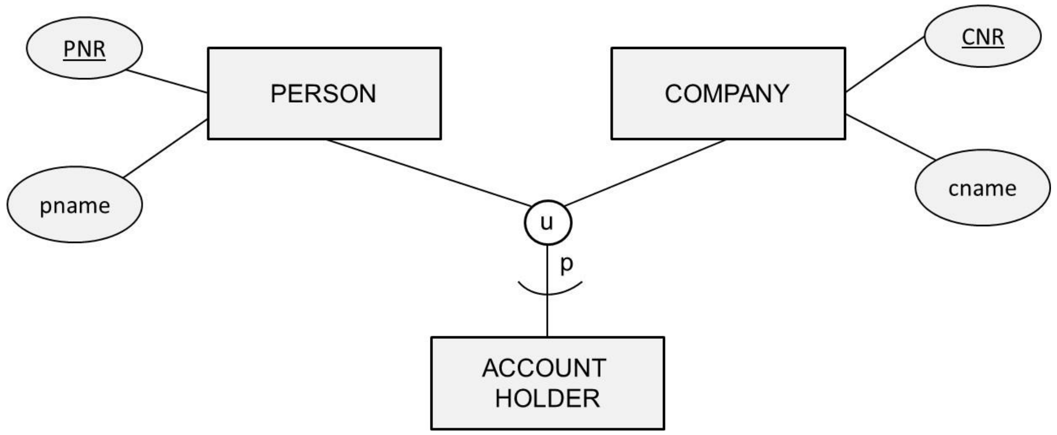
Categorization

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



Categorization

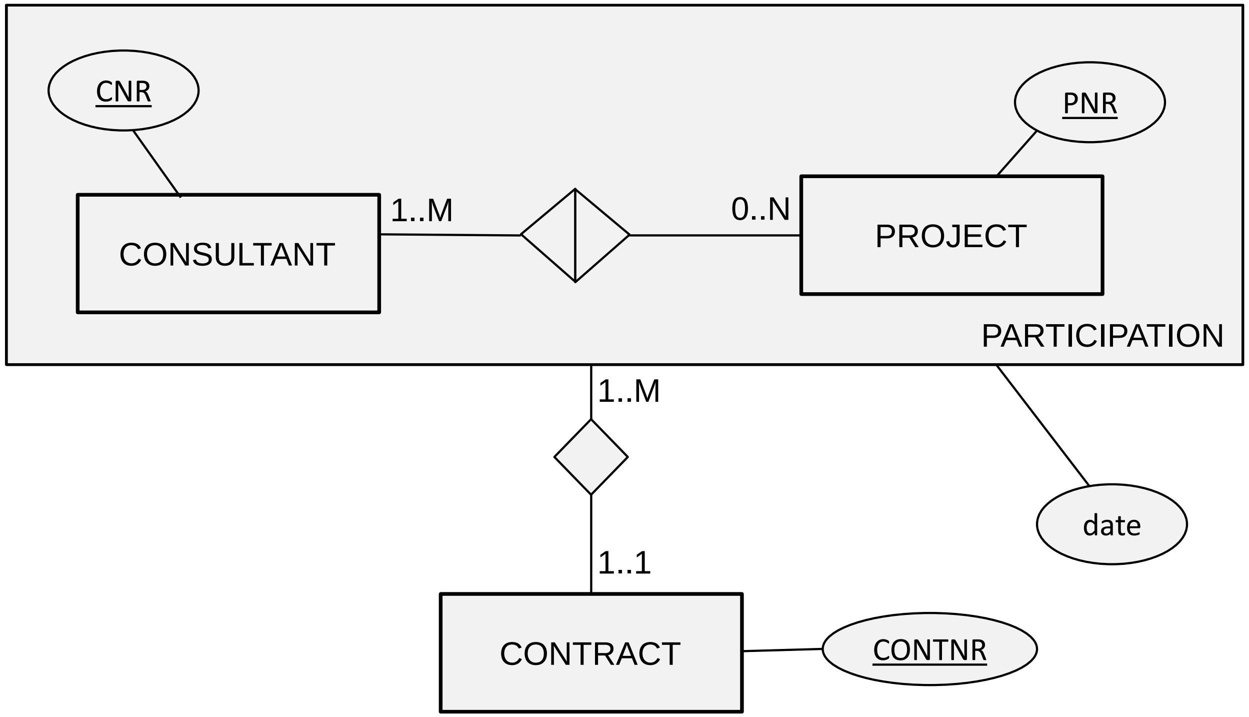
- Inheritance in the case of categorization corresponds to an entity inheriting only the attributes and relationships of that superclass it is a member of (selective inheritance)
- A categorization can be total or partial
  - Total: all entities of the superclasses belong to the subclass
  - Partial: not all entities of the superclasses belong to the subclass



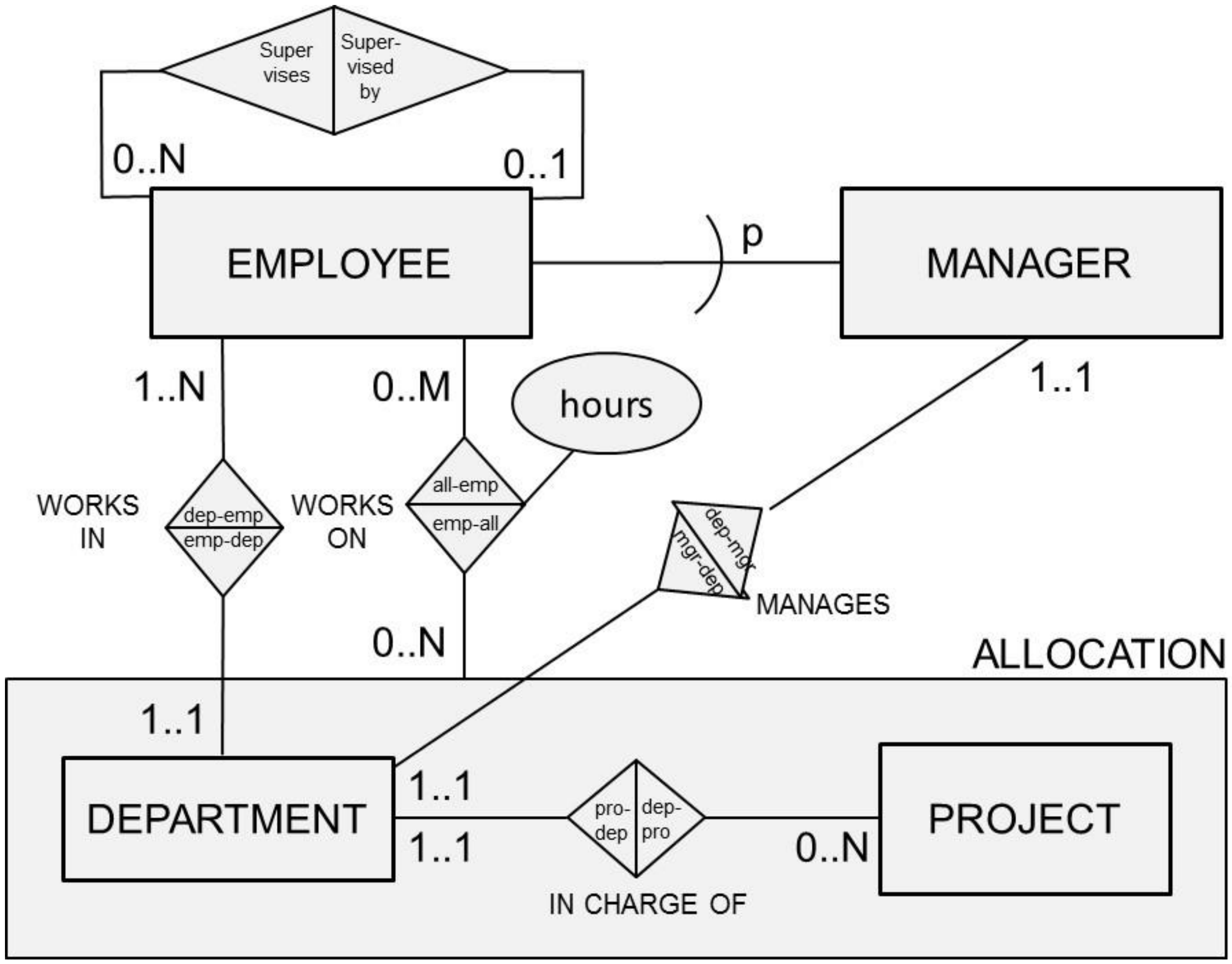
Note: total categorization can also be represented as a specialization/generalization!

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



Examples of the EER Model



Designing the EER Model

1. Identify the entity types
2. Identify the relationship types and assert their degree
3. Assert the cardinality ratios and participation constraints (total versus 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

UML Class Diagram

- Origin
- Recap of Object Orientation
- Classes
- Variables
- Access Modifiers
- Associations
- Specialization/Generalization
- Aggregation
- UML Class Diagram Example
- Advanced UML Modeling Concepts
- UML Class Diagram versus EER

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

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

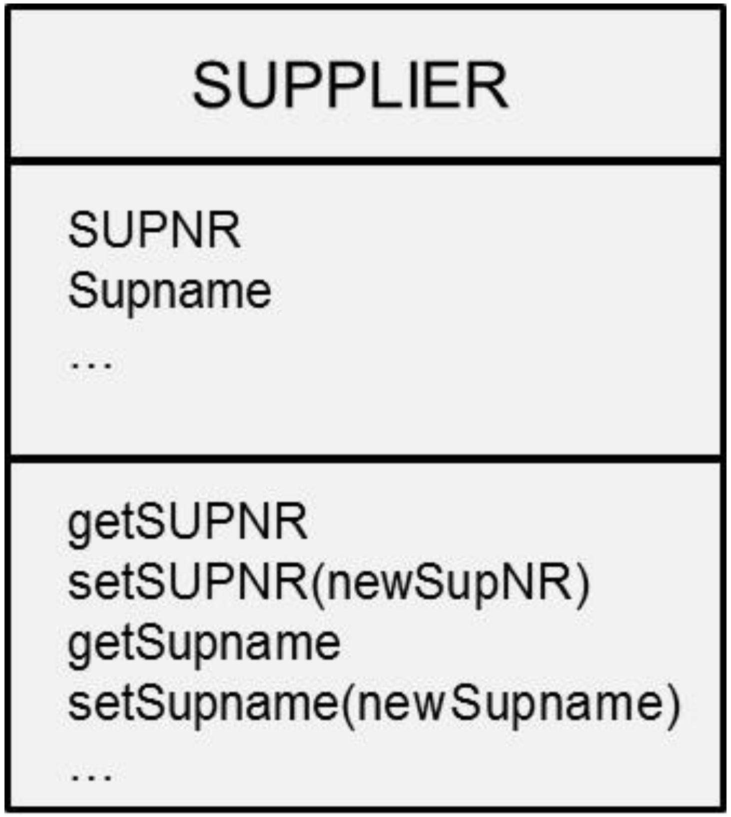
Recap of Object Orientation

- Example class Student
  - Objects: Bart, Wilfried, Seppe
  - Example variables: student’s name, gender and birthdate
  - Example methods: calcAge, isBirthday, hasPassed(courseID)
- Information hiding (a.k.a. encapsulation) states that the variables of an object can only be accessed through either getter or setter methods
  - getter method is used to retrieve the value of a variable
  - setter method assigns a value to a it

Recap of Object Orientation

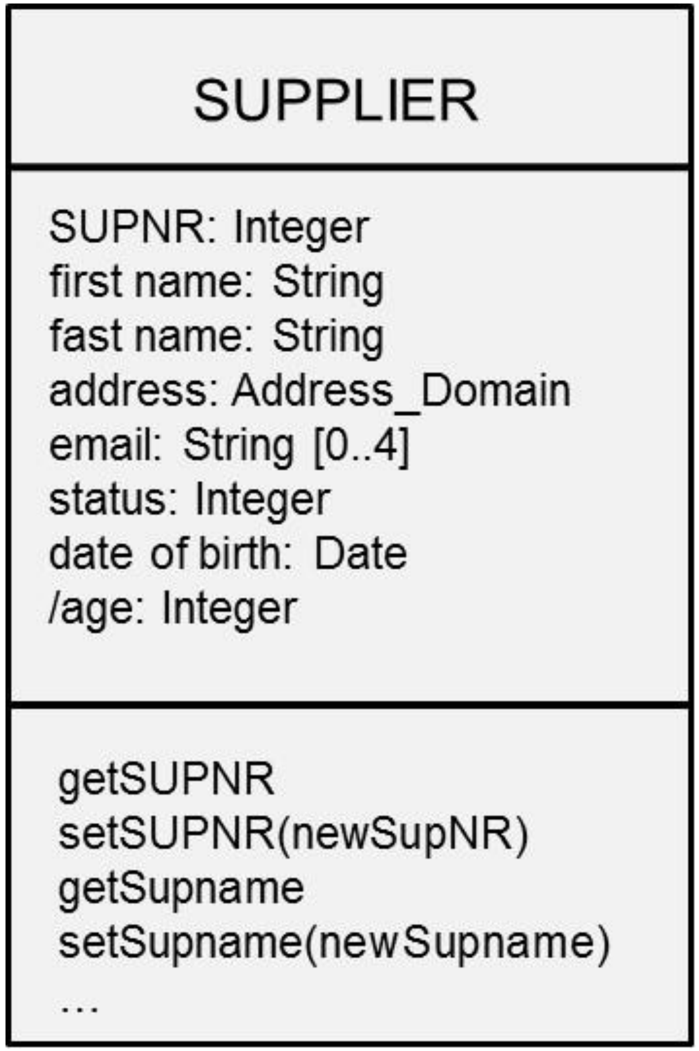
- Inheritance
  - A superclass can have one or more subclasses which inherit both the variables and methods from the superclass
- Method overloading
  - Various methods in the same class can have the same name, but a different number or type of input arguments

Classes


SUPPLIER
SUPNR
Supname
...
getSUPNR
setSUPNR(newSupNR)
getSupname
setSupname(newSupname)
...

Variables

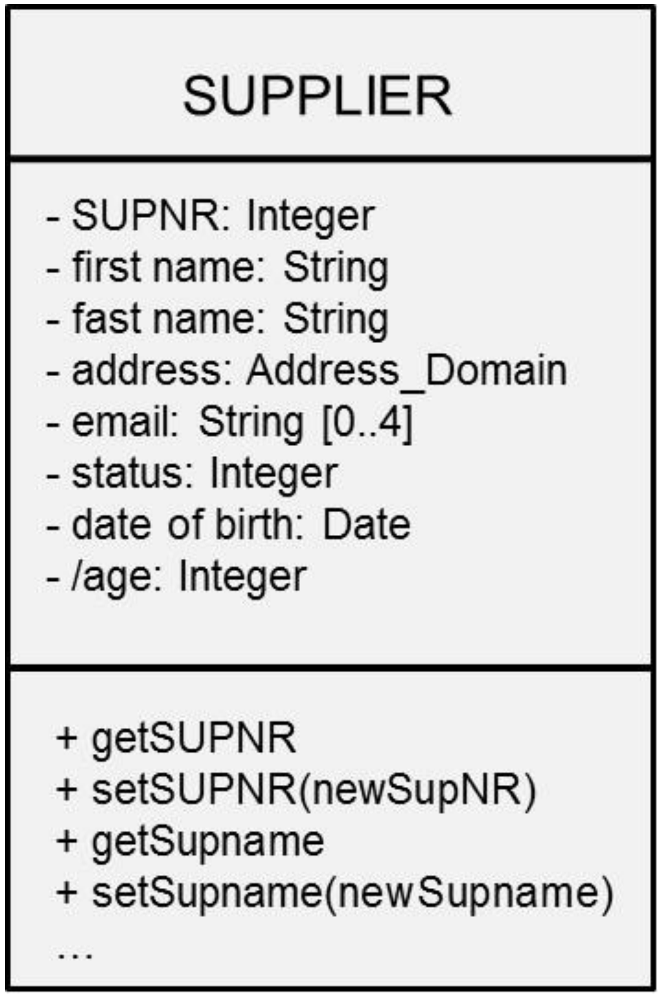
- Variables with unique values (~ key attribute types in ER) are not directly supported in UML
- UML provides a set of primitive types such as String, Integer, and Boolean
- It is also possible to define your own data types or domains
- Composite/Multi-valued/Derived variables



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)

Access Modifiers



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 the ER model)

Associations

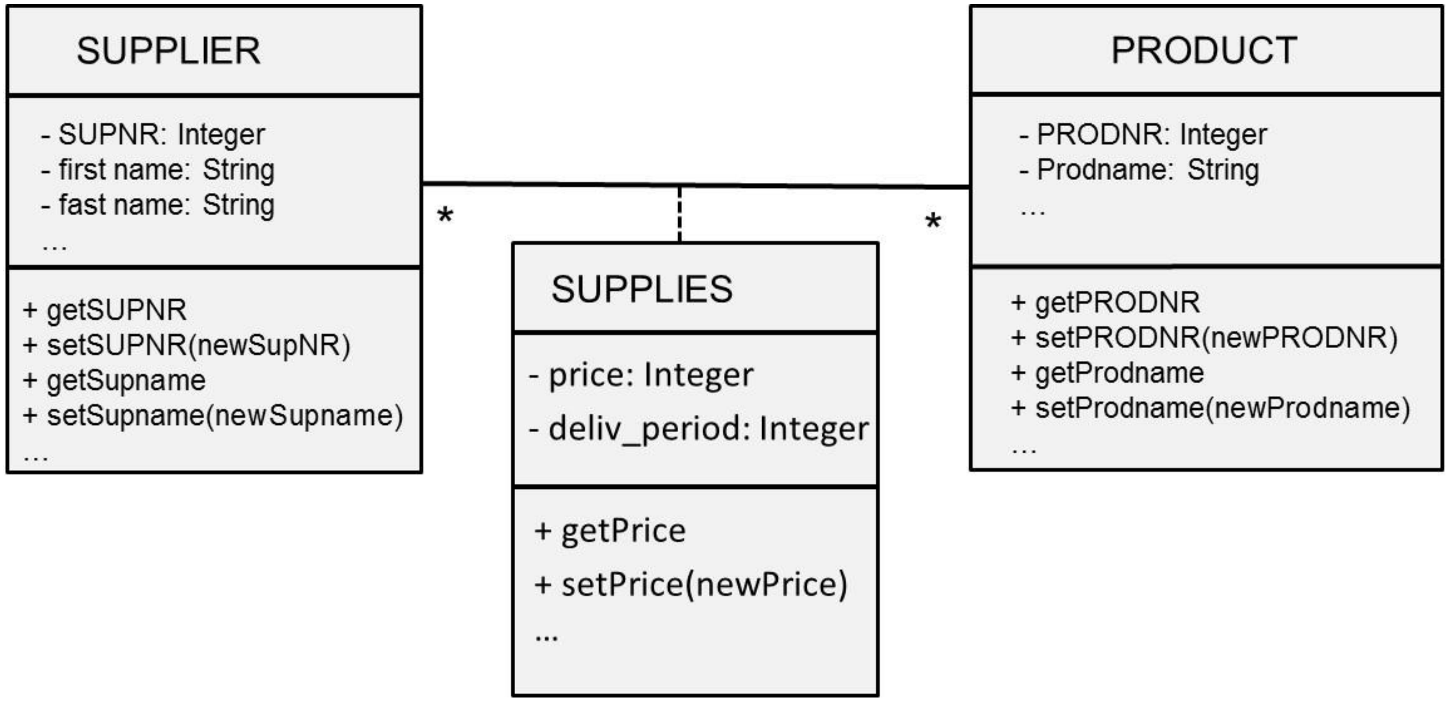
UML class diagram multiplicity	**ER model cardinality**
*	0..N
0..1	0..1
1..*	1..N
1	1..1

Associations

- Association Class
- Unidirectional versus Bidirectional Association
- Qualified Association

Association Class

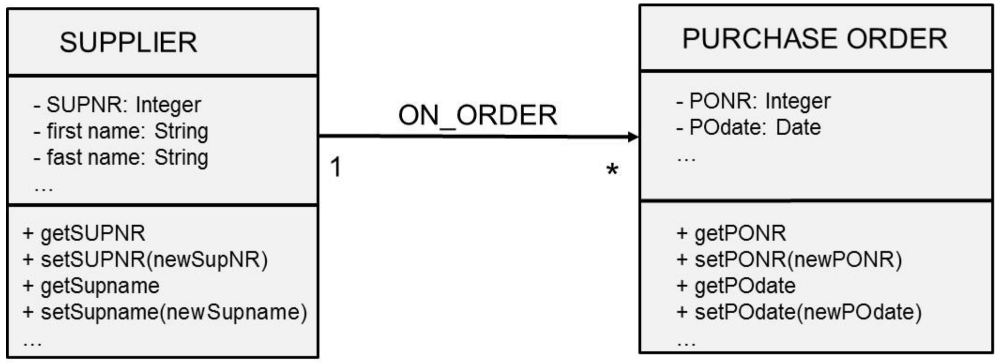
- In case an association has variables and/or methods on its own, it can be modelled as an association class



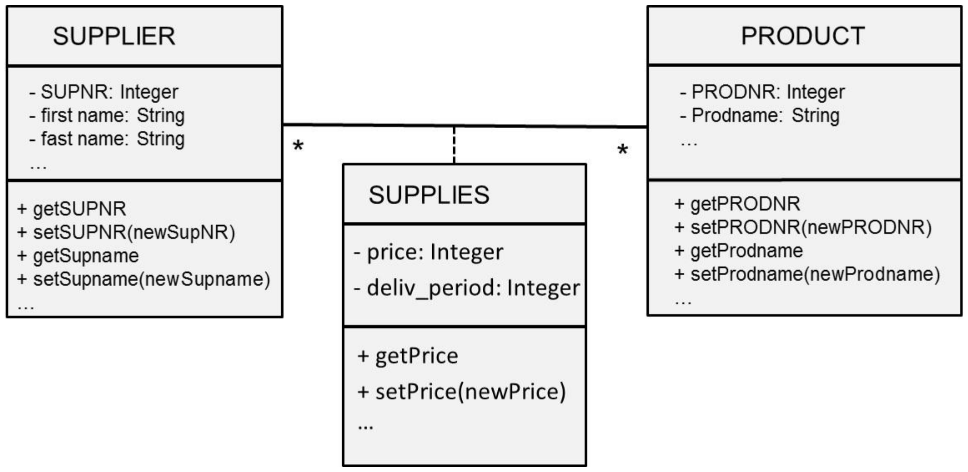
Unidirectional versus Bidirectional Association

- Associations can be augmented with direction reading arrows, which specify the direction of querying or navigating through it

Unidirectional



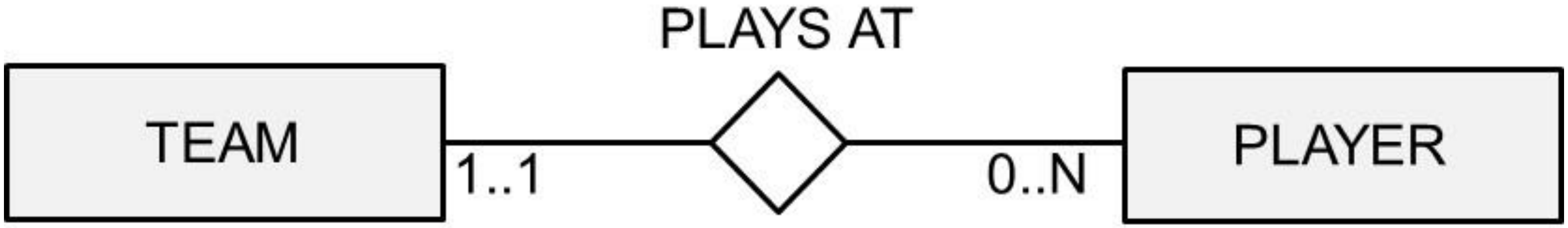
Bidirectional

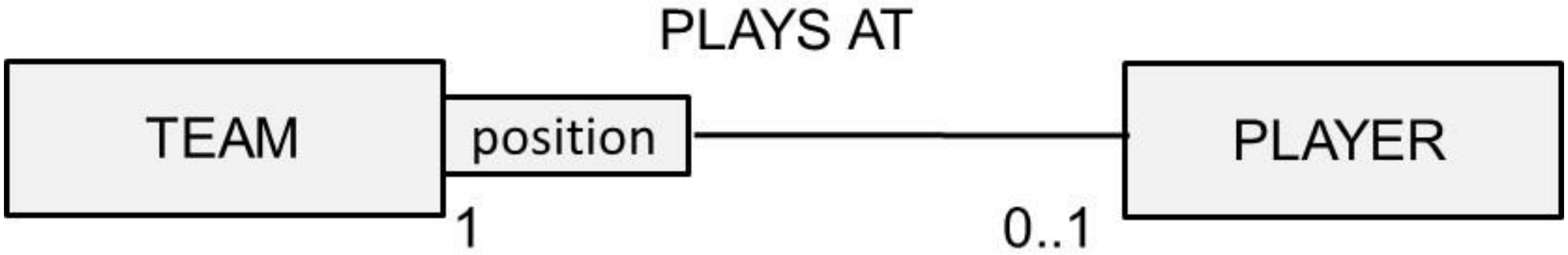


Qualified Association

- A qualified association is a special type of association that uses a qualifier to further refine the association
- The qualifier specifies one or more variables that are used as index key for navigating from the qualified class to the target class
  - reduces the multiplicity of the association because of this extra key

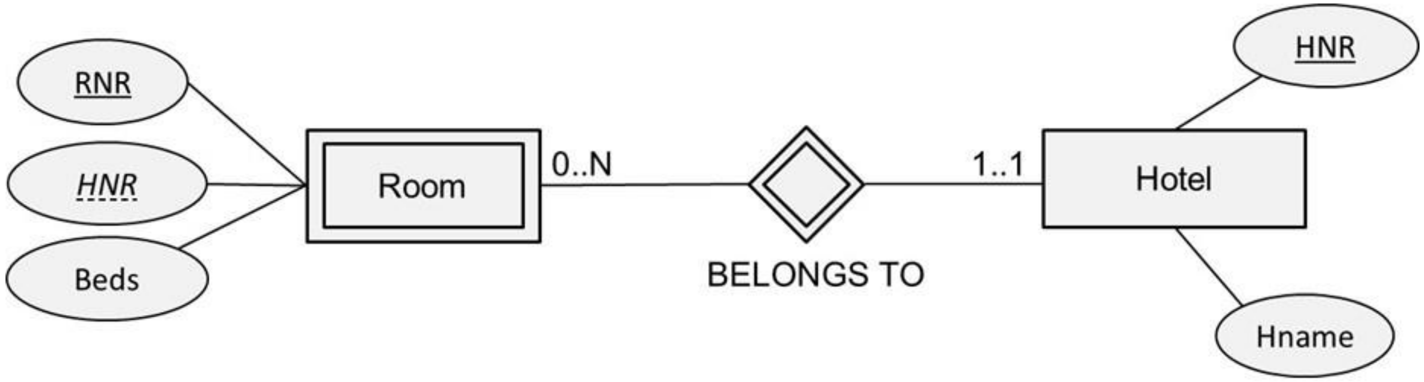
Qualified Association

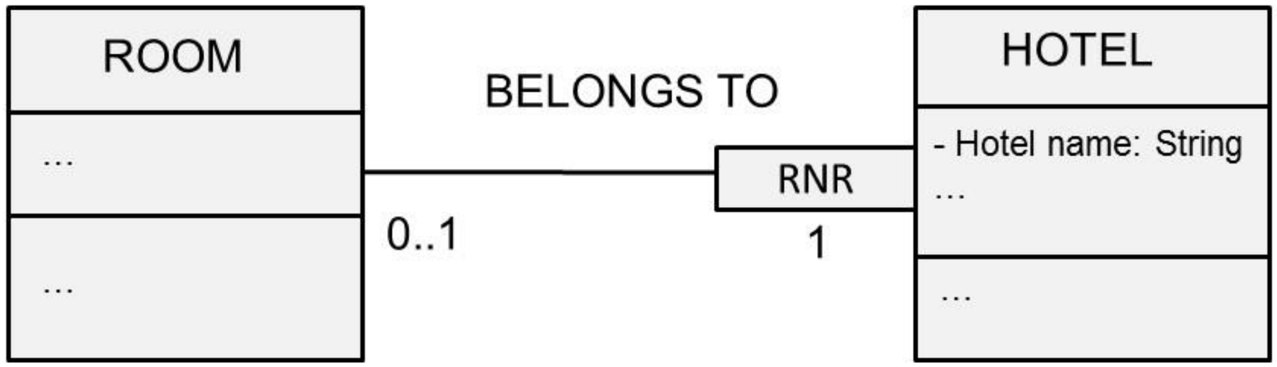




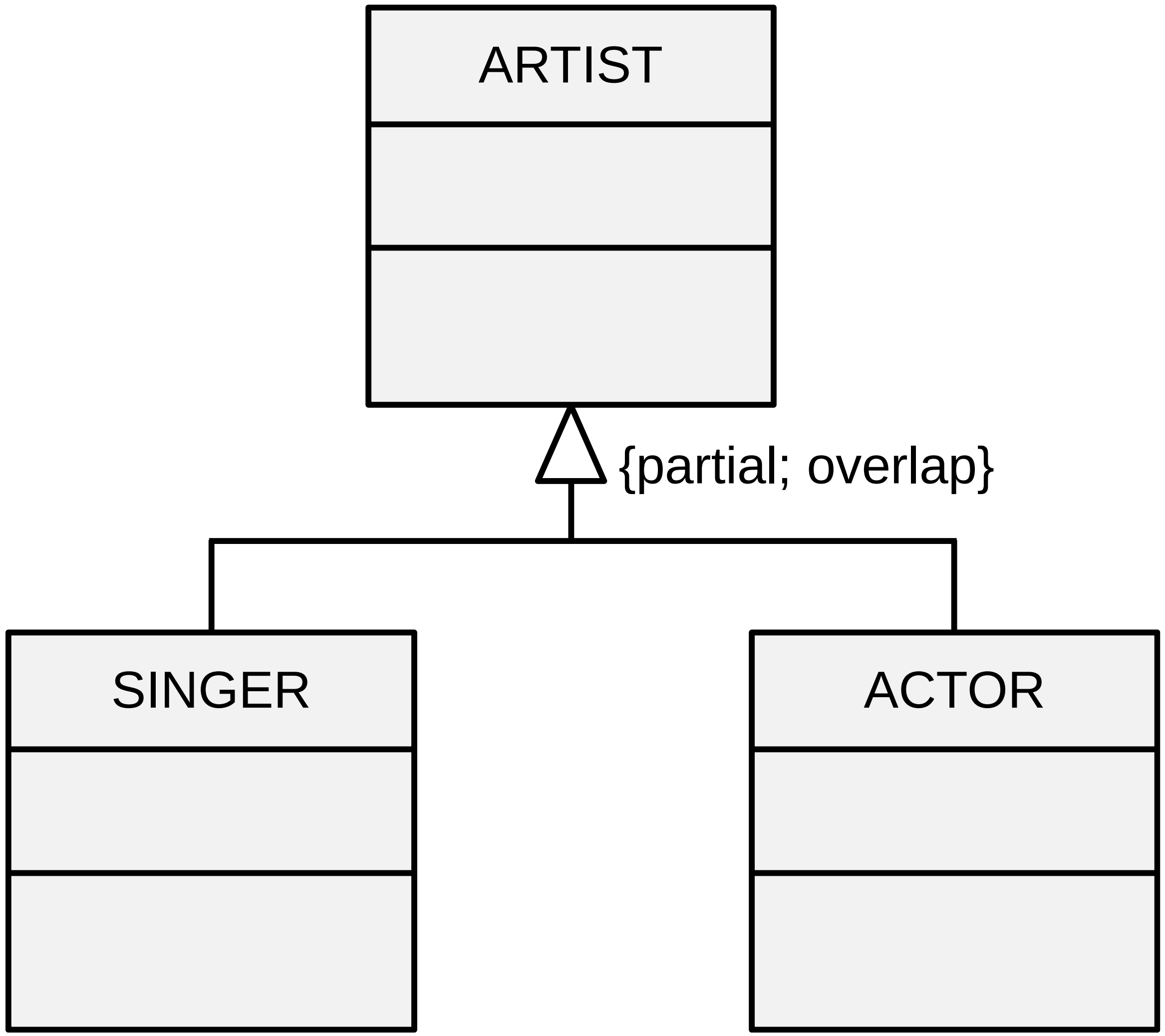
Qualified Association

- Qualified associations can be used to represent weak entity types





Specialization/Generalization



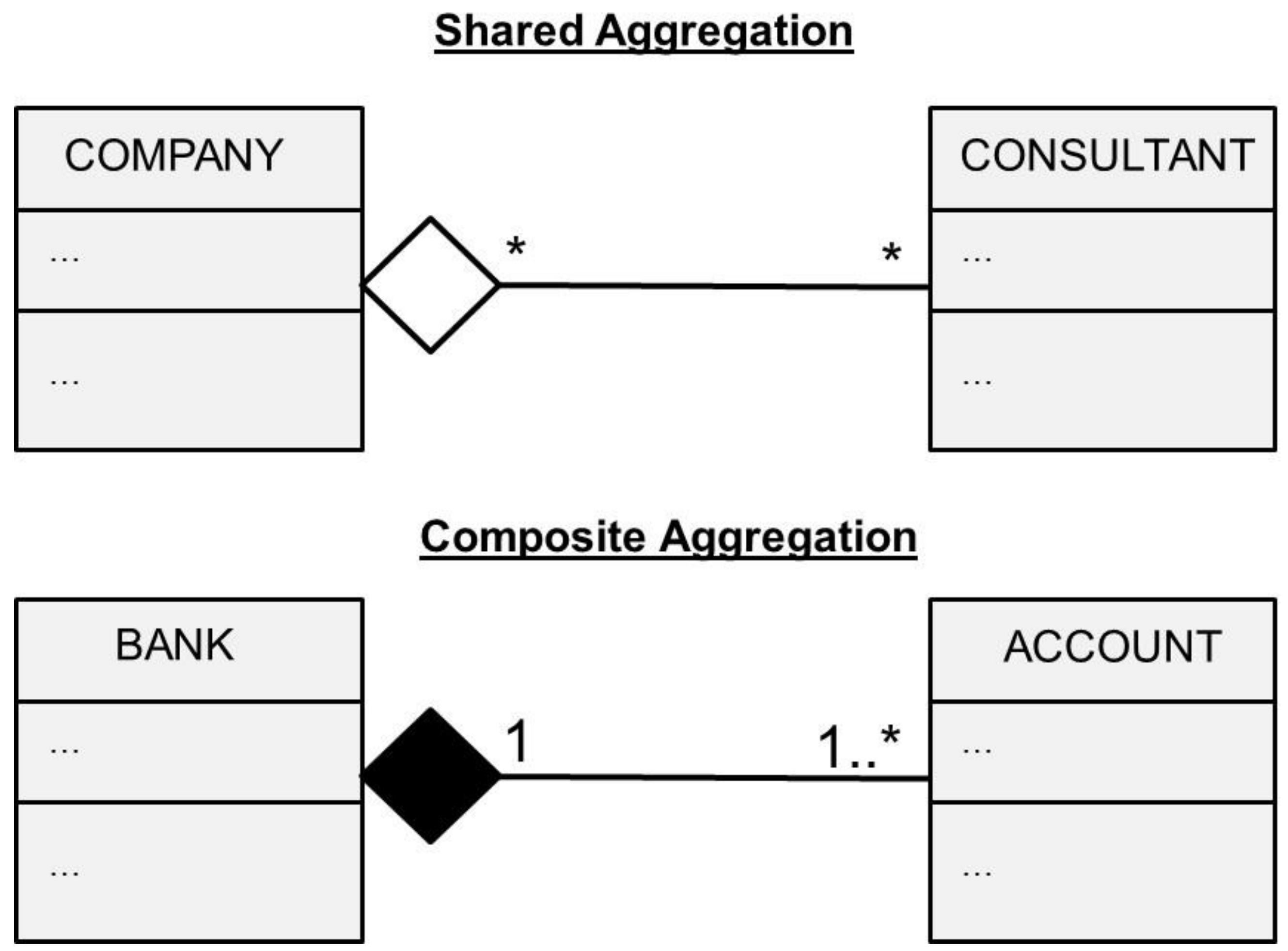
Aggregation

- Aggregation represents a composite to part relationship whereby a composite class contains a part class
- Two types in UML: shared and composite aggregation
- Shared aggregation (a.k.a. aggregation)
  - part object can simultaneously belong to multiple composite objects
  - maximum multiplicity at the composite side is undetermined
  - part object can also occur without belonging to a composite object
  - loose coupling

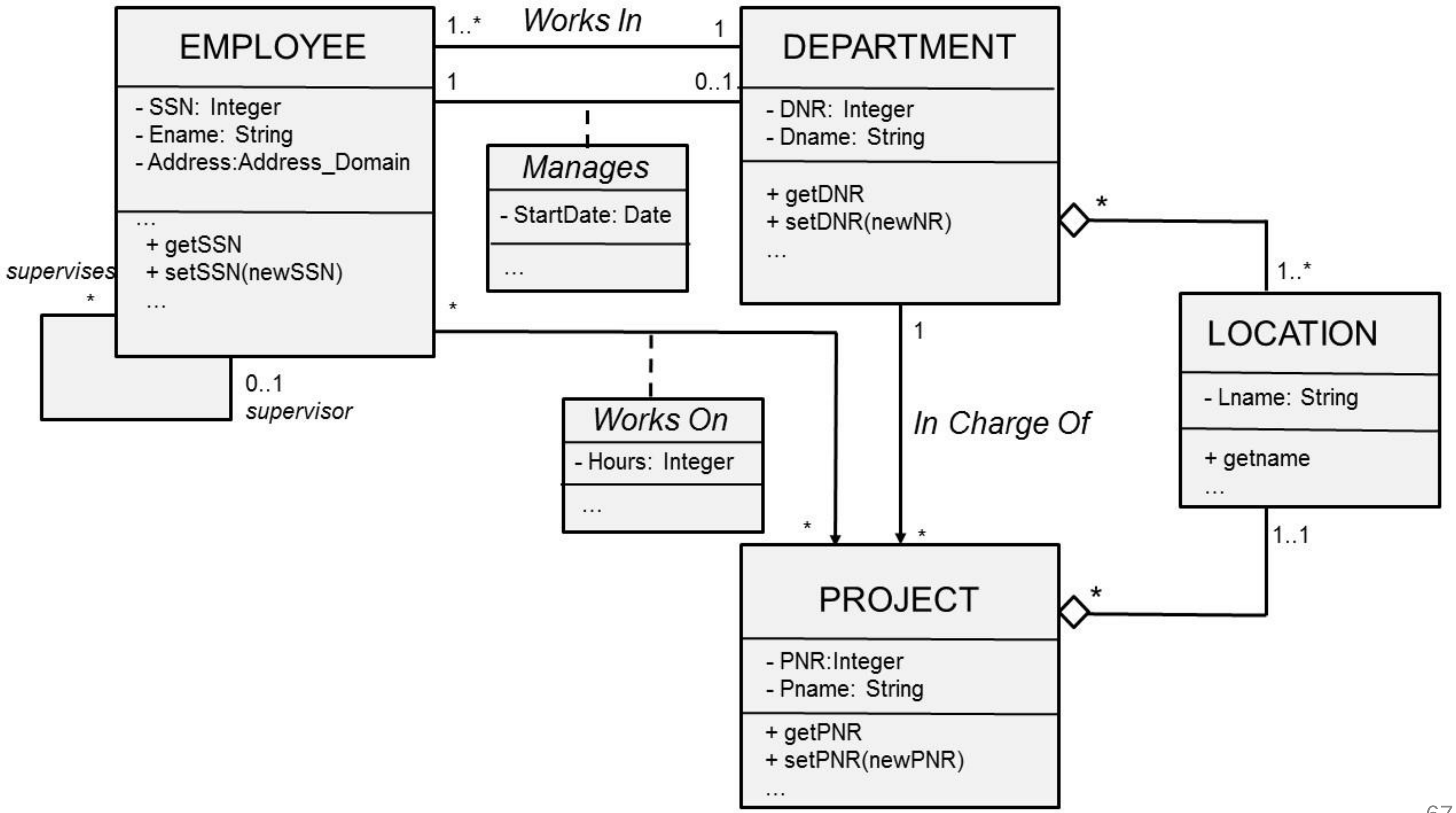
Aggregation

- Composite aggregation (a.k.a. composition)
  – the part object can only belong to one composite
  – maximum multiplicity at the composite side is 1
  – minimum multiplicity can be either 1 or 0
  – tight coupling

Aggregation



UML Example

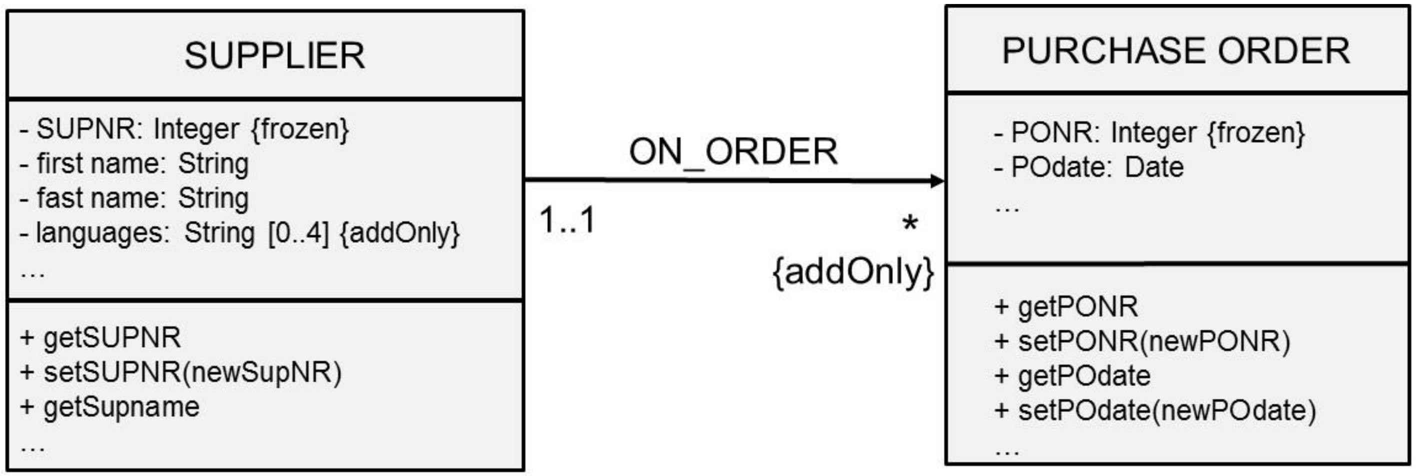

EMPLOYEE
- SSN: Integer
- Ename: String
- Address:Address_Domain
…
+ getSSN
+ setSSN(newSSN)
…
DEPARTMENT
- DNR: Integer
- Dname: String
+ getDNR
+ setDNR(newNR)
…
Works In
1..*
1
1
0..1
Manages
- StartDate: Date
…
supervises
*
0..1
supervisor
*
Works On
- Hours: Integer
…
*
1
In Charge Of
*
*
1..*
LOCATION
- Lname: String
+ getname
…
1..1
*
PROJECT
- PNR:Integer
- Pname: String
+ getPNR
+ setPNR(newPNR)
…

Advanced UML Modeling Concepts

- Changeability property
- Object Constraint Language (OCL)
- Dependency relationship

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



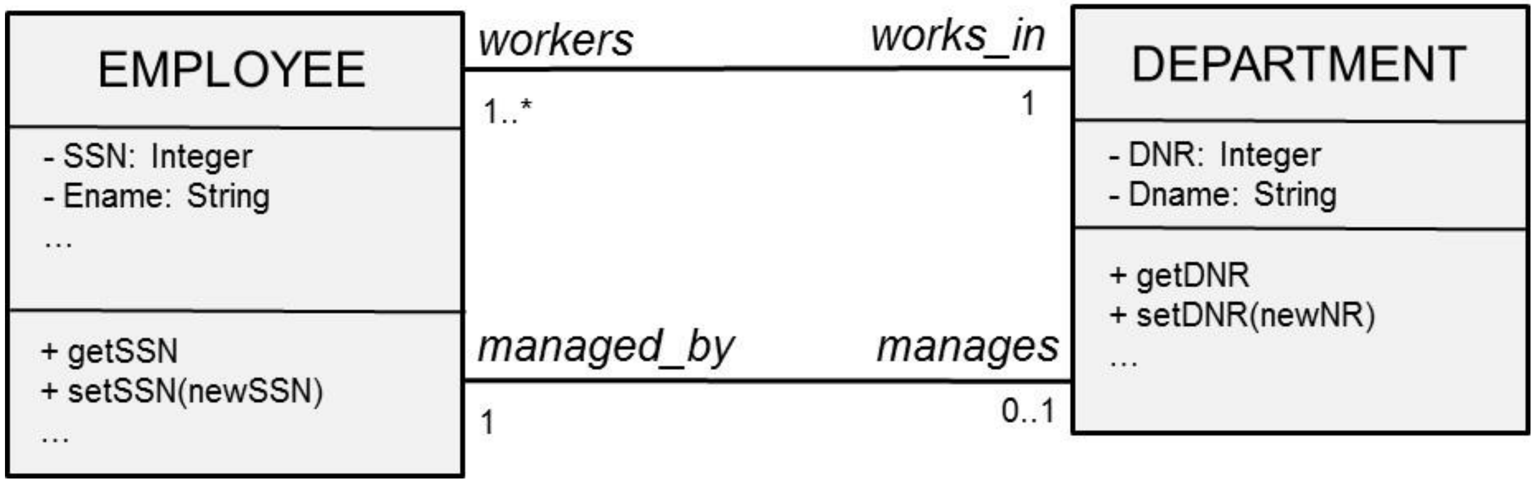
Object Constraint Language (OCL)

- The Object Constraint Language (OCL) can be used to specify various types of constraints in a declarative way
  - no control flow or procedural code is provided
  - 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/

Object Constraint Language (OCL)

- A class invariant is a constraint which holds for all objects of a class
  - Example: SUPPLIER: SUPSTATUS>100
- Pre- and post-conditions on methods must be true when a method either begins or ends
  - Example: before the method withdrawal can be executed, the balance must be positive. After it has been executed, the balance must still be positive.

Object Constraint Language (OCL)

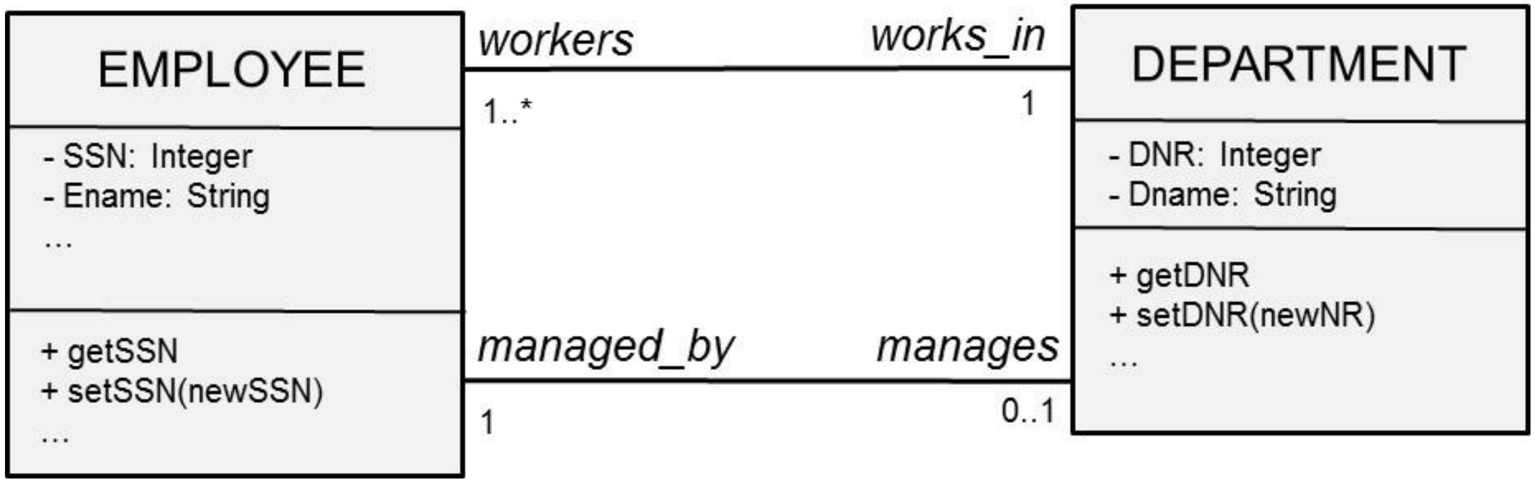


- Constraint: manager of a department should be at least 10 years employed

  `Context: Department`

  `invariant: self.managed_by.yearsemployed>10`

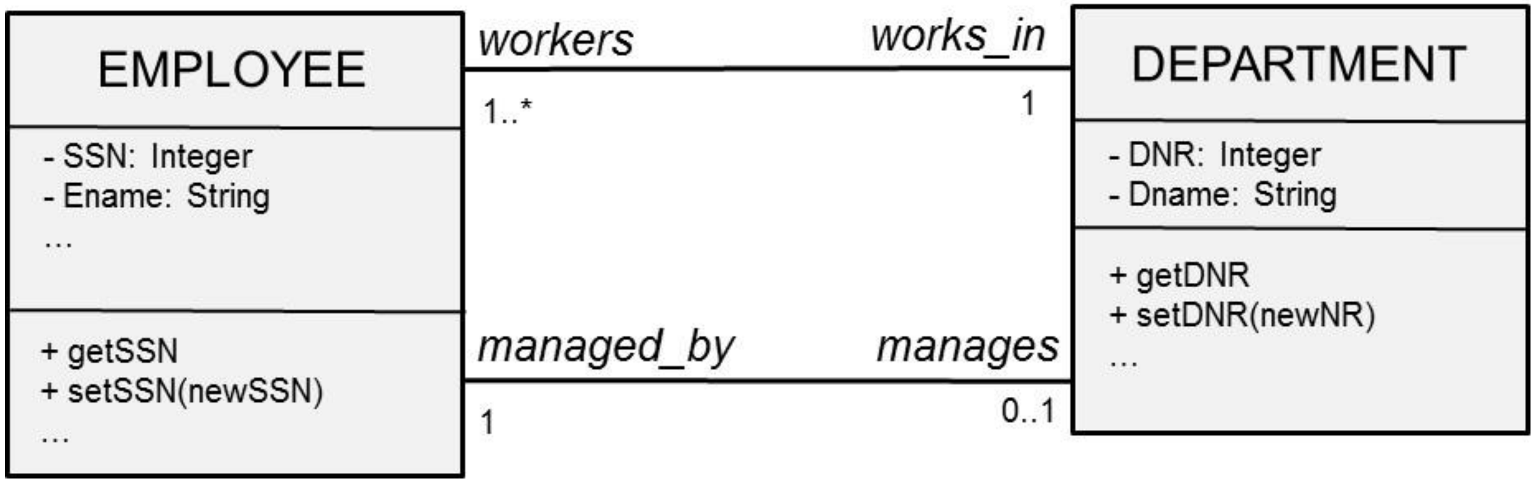
Object Constraint Language (OCL)



- A department should have at least 20 employees.

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

Object Constraint Language (OCL)

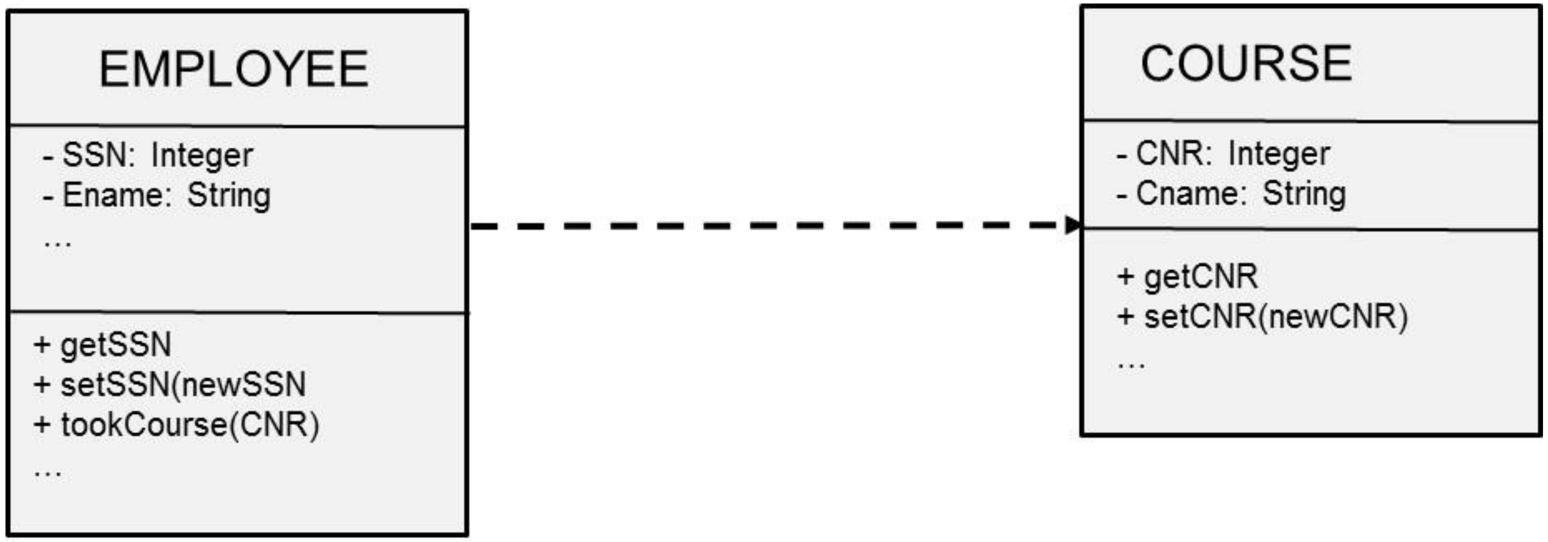


- Constraint: A manager of a department must also work in the department

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

Dependency Relationship

- 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



UML Class Diagram versus EER

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

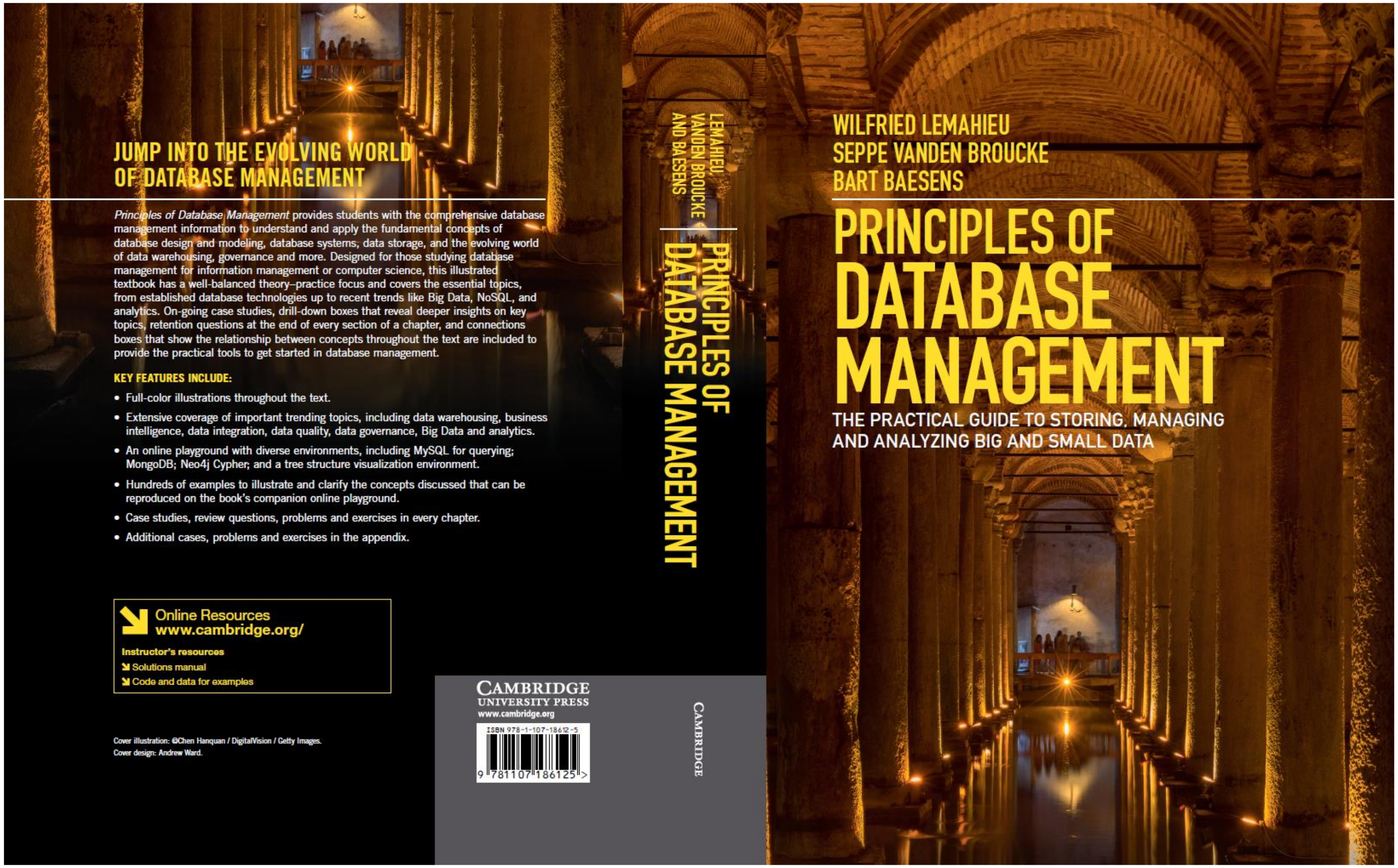
UML versus EER

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

Conclusions

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

More information?



JUMP INTO THE EVOLVING WORLD OF DATABASE MANAGEMENT

Principles of Database Management provides students with the comprehensive database management information to understand and apply the fundamental concepts of database design and modeling, database systems, data storage, and the evolving world of data warehousing, governance and more. Designed for those studying database management for information management or computer science, this illustrated textbook has a well-balanced theory–practice focus and covers the essential topics, from established database technologies up to recent trends like Big Data, NoSQL, and analytics. On-going case studies, drill-down boxes that reveal deeper insights on key topics, retention questions at the end of every section of a chapter, and connections boxes that show the relationship between concepts throughout the text are included to provide the practical tools to get started in database management.

KEY FEATURES INCLUDE:

- Full-color illustrations throughout the text.
- Extensive coverage of important trending topics, including data warehousing, business intelligence, data integration, data quality, data governance, Big Data and analytics.
- An online playground with diverse environments, including MySQL for querying; MongoDB; Neo4j Cypher; and a tree structure visualization environment.
- Hundreds of examples to illustrate and clarify the concepts discussed that can be reproduced on the book's companion online playground.
- Case studies, review questions, problems and exercises in every chapter.
- Additional cases, problems and exercises in the appendix.

Online Resources
www.cambridge.org/

Instructor's resources
Solutions manual
Code and data for examples

Cover illustration: ©Chen Hanquan / DigitalVision / Getty Images.
Cover design: Andrew Ward.

CAMBRIDGE
UNIVERSITY PRESS
www.cambridge.org

ISBN 978-1-107-18612-5

LEMAHIEU
VANDEN BROUCKE
AND BAESENS

PRINCIPLES OF DATABASE MANAGEMENT

CAMBRIDGE

WILFRIED LEMAHIEU
SEPPE VANDEN BROUCKE
BART BAESENS

PRINCIPLES OF DATABASE MANAGEMENT

THE PRACTICAL GUIDE TO STORING, MANAGING AND ANALYZING BIG AND SMALL DATA

www.pdbmbook.com