[F000923A] Databasesystemen

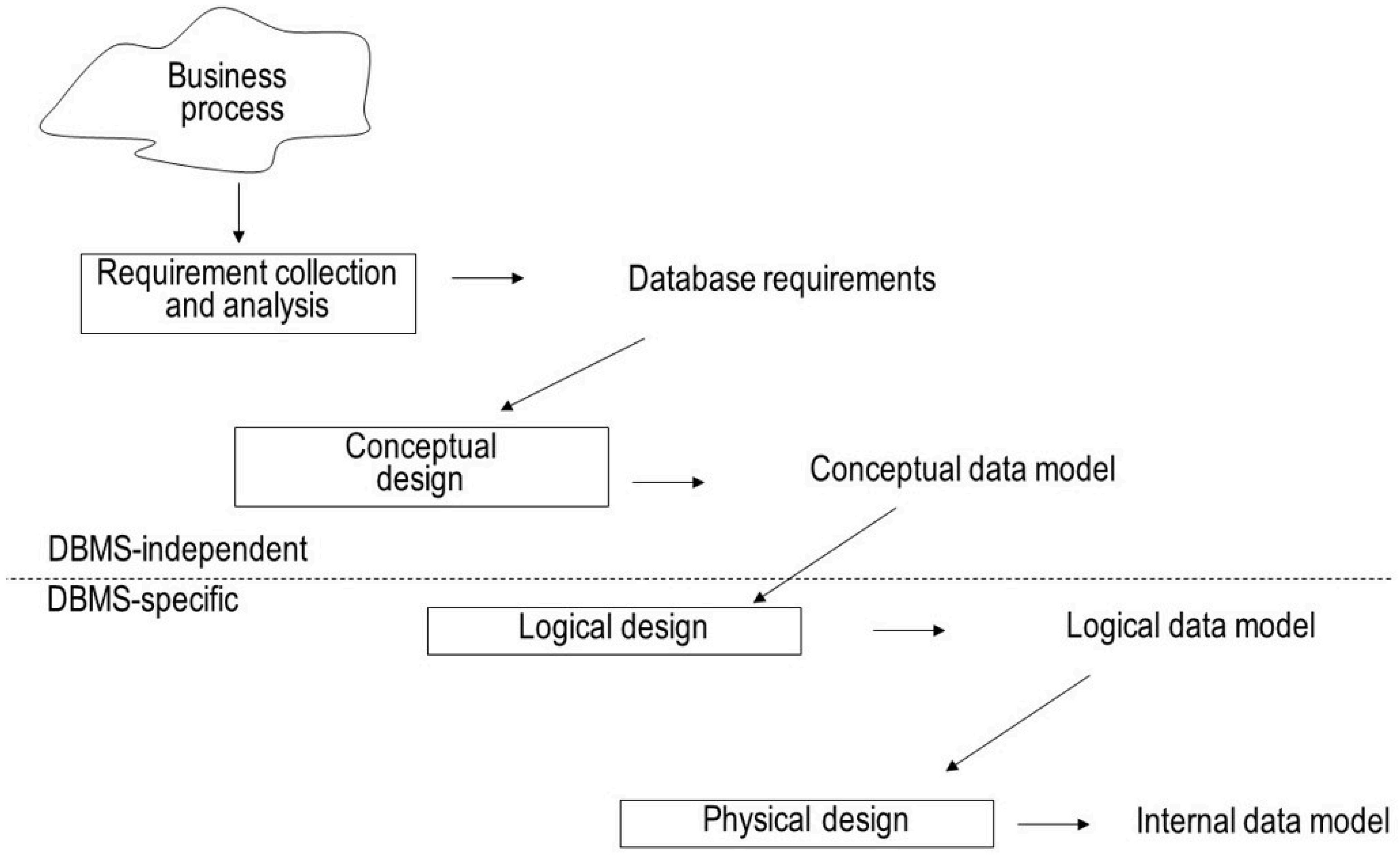
Conceptual Modeling

Overview

- Phases of database design
- Entity Relationship (ER) model
- Enhanced Entity Relationship (EER) model
- UML class diagram

→ See Chapter 3 in book

Phases of database design



Phases of database design

Designing a database is a *multi-step process*:

- Starts from a **business process** or set of processes that needs to be supported
  - E.g. procurement application, invoice handling process, logistics process, salary administration
- First step is **requirement collection and analysis** to carefully understand the different steps and data needs of the process
  - The information architect will collaborate with the business user to elucidate the database requirements
  - Various techniques can be used: interviews or surveys with end users, inspections of the documents which are used in the current process, informal meetings, etc.

Phases of database design

- During the **conceptual design** phase, parties try to formalize the data requirements in a conceptual data model
  - This should be a high-level model: both easy to understand to the business user and formal enough to the database designer who will use it in a next step
  - The conceptual data model must be user-friendly, and preferably have a graphical representation such that it can be used as a communication and discussion instrument
  - It should be flexible such that new or changing data requirements can 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
  - Obviously, this conceptual model will also have its limitations which should be clearly documented and followed up during application development

Phases of database design

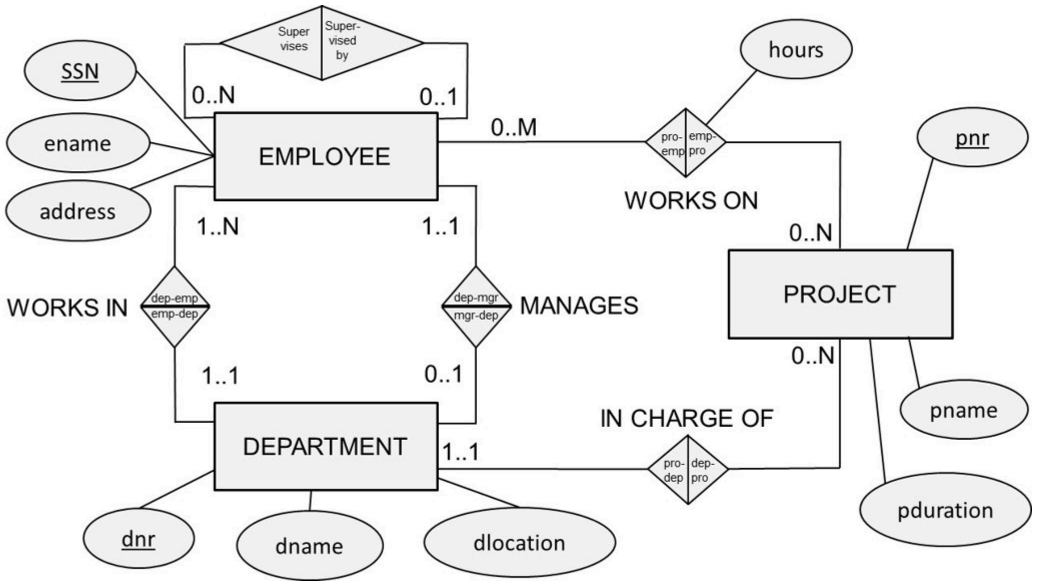
- Once all parties have agreed upon the conceptual data model, it can be mapped to a **logical data model** by the database designer during the logical design step
    - The logical data model is based upon the data model used by the implementation environment
    - Although at this stage it is already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

Phases of database design

- In a final 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 DDL is generated and the data 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 mapping step should be documented and followed-up

Entity Relationship (ER) model

- The Entity Relationship (ER) model was introduced and formalized by Peter Chen in 1976
- One of the most popular data models for **conceptual data modeling**
- The ER model has a **graphical notation**:



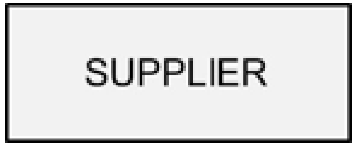
Entity Relationship (ER) model

- Entity types
- Attribute types
- Relationship types
- Weak entity types
- Ternary relationship types
- Examples of the ER model
- Limitations of the ER model

Entity Relationship (ER) model

Entity types

- An **entity type** represents a business concept with an unambiguous meaning to a particular set of users
    - Examples: supplier, student, product or employee
    - Drawn as a box



SUPPLIER

- An **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
    - Not shown in the model (since this relates to the database state)

Entity Relationship (ER) model

Entity types

Note: “SUPPLIER”, “Supplier”, “supplier” are all ok - no strict semantics but be consistent!

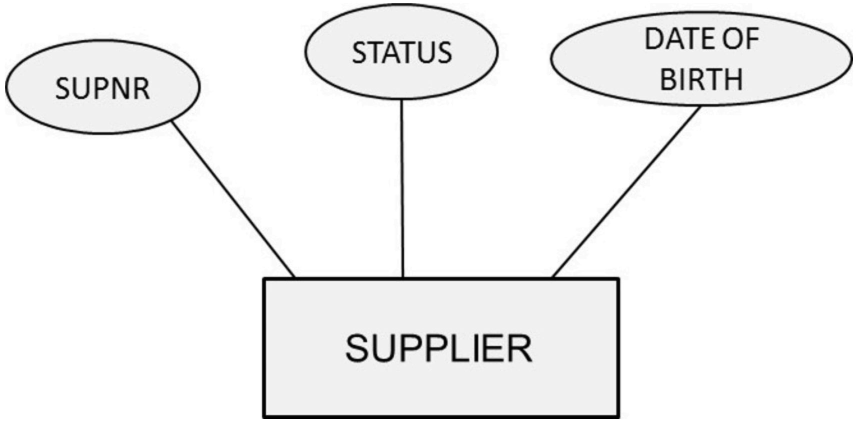
The entity is the main building block of this any many other modeling language!

- Architectural challenge: in case of doubt: “perhaps this should be its own entity”

Entity Relationship (ER) model

Attribute types

- An **attribute type** represents a property of an entity type
  - Example: name and address are attribute types of the entity type supplier
  - Drawn as circles connected to entity types



- An **attribute** is an instance of an attribute type
  - Not shown in the model

Entity Relationship (ER) model

Attribute types

- Domains
- Key attribute types
- Simple versus composite attribute types
- Single-valued versus multi-valued attribute types
- Derived attribute types
- Final remarks

Entity Relationship (ER) model

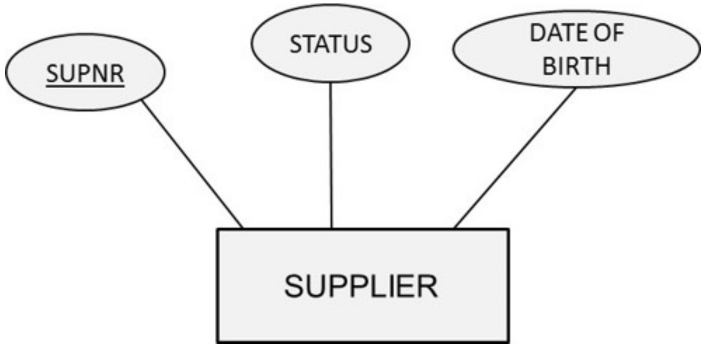
Attribute types: domains

- A **domain** specifies the set of values that may be assigned to an attribute for each individual entity
  - Example: gender: [‘M’, ‘F’, ‘X’]; age: [0-100]
- A domain can also optionally contain NULL values
  - NULL value: value is not known, not applicable or not relevant
  - Not the same as 0 or an empty string of text!
- Domains are not displayed visually in ER model itself
  - Need to be written down in documentation
  - Or you could add it in the circles yourself, but this makes the model harder to read at a glance

Entity Relationship (ER) model

Attribute types: key attribute types

- A **key attribute type** is an attribute type whose values are distinct (unique) 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
- Key attribute types are underlined in the ER model



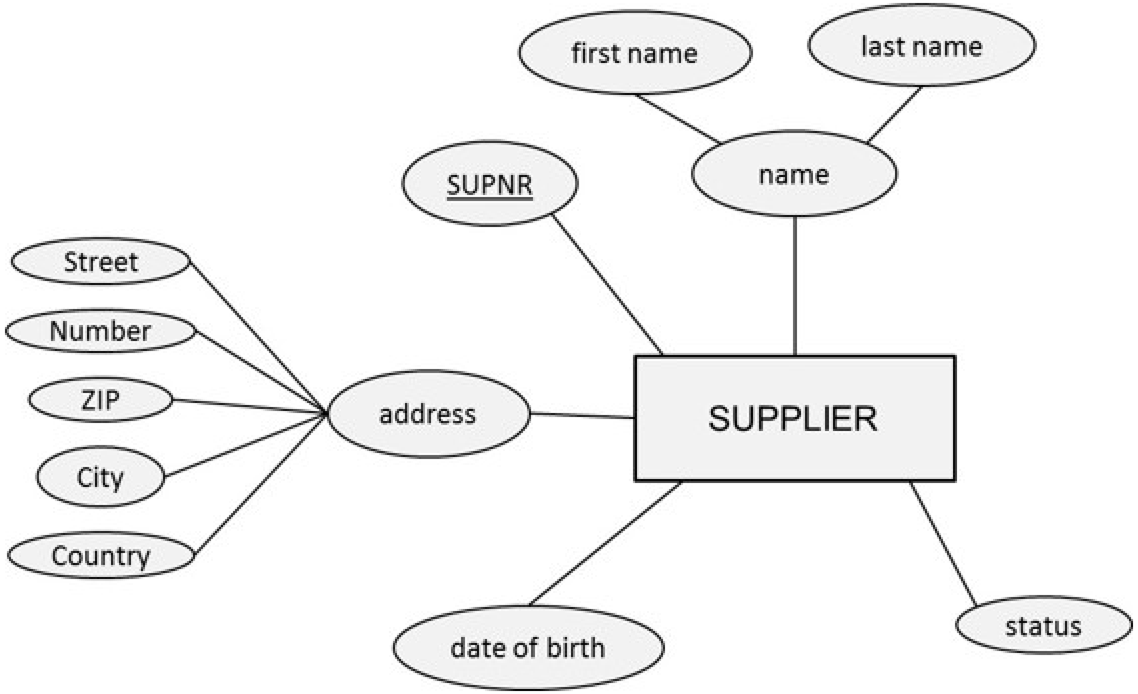
Entity Relationship (ER) model

Attribute types: 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 (street, number, postal code), name (first and last name)
- Shown by connecting types together in the ER model

Entity Relationship (ER) model

Attribute types: simple versus composite attribute types



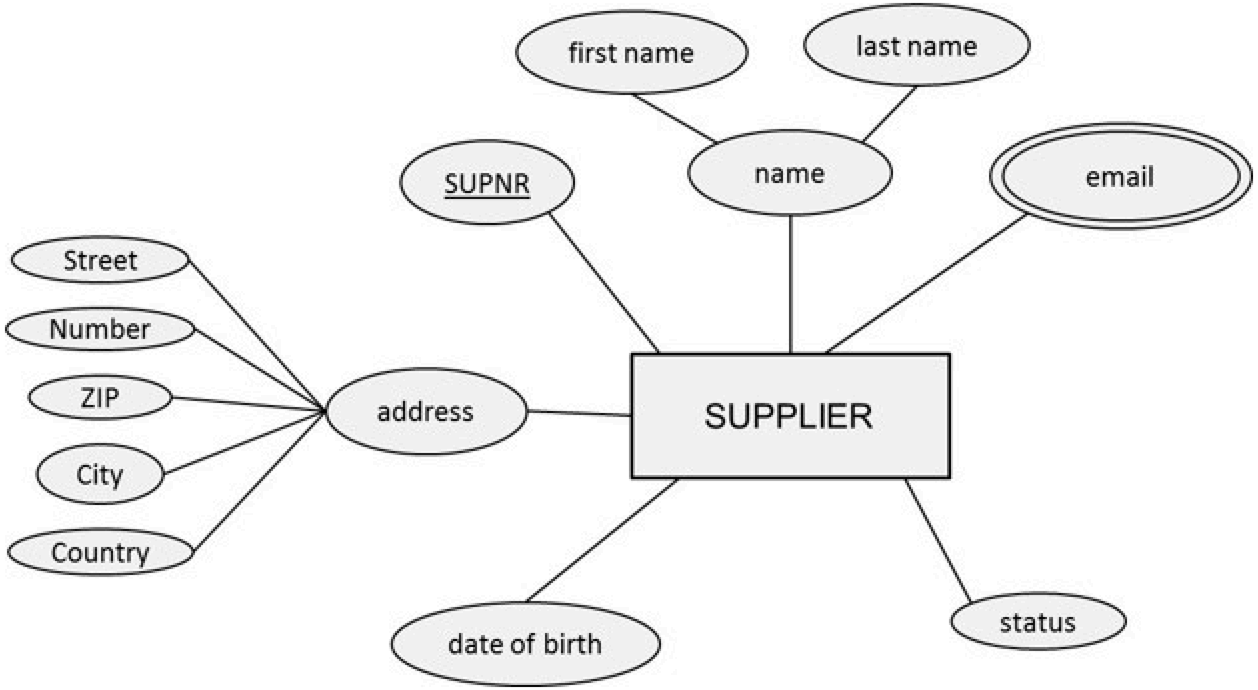
Entity Relationship (ER) model

Attribute types: 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
- Shown with a double border in the ER model

Entity Relationship (ER) model

Attribute types: single-valued versus multi-valued attribute types



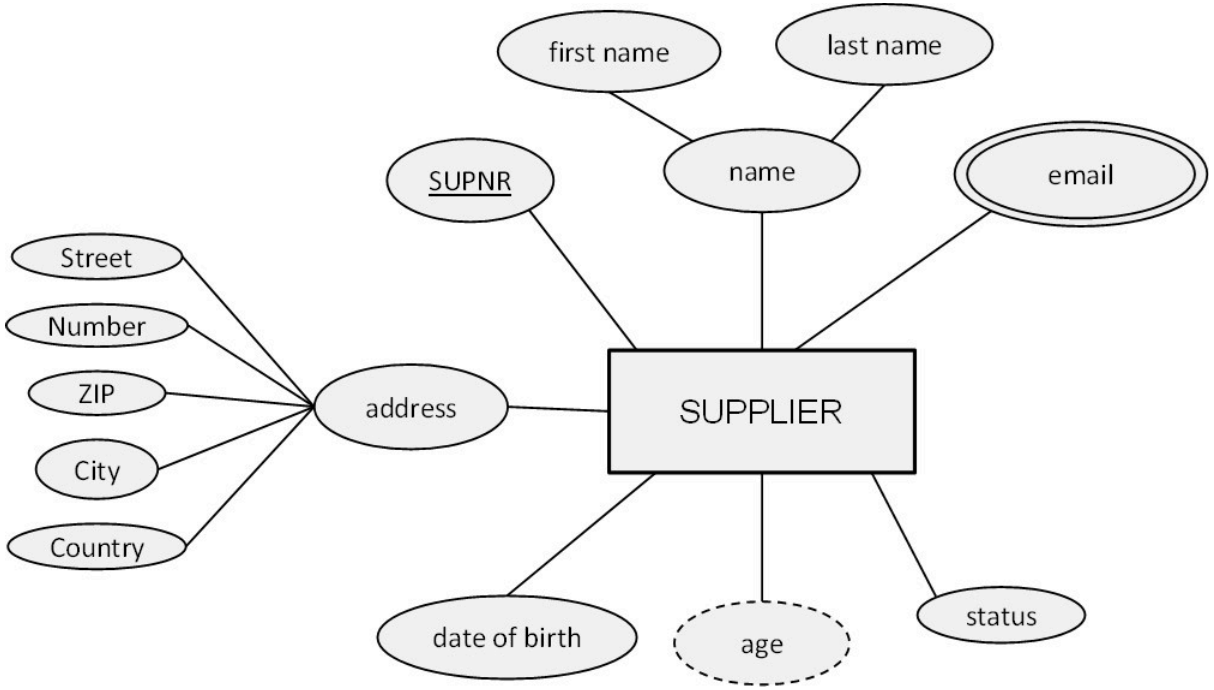
Entity Relationship (ER) model

Attribute types: derived attribute types

- A **derived attribute type** is an attribute type which can be derived from another attribute type
- Shown with a dotted border in the ER model

Entity Relationship (ER) model

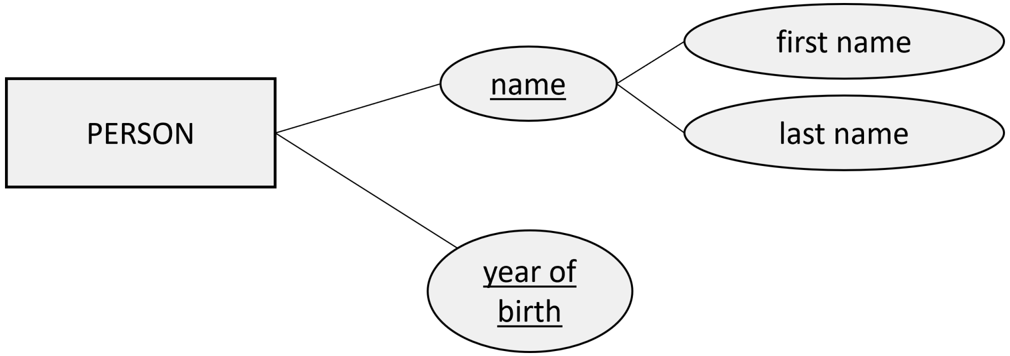
Attribute types: derived attribute types



- Age can be derived (calculated) from date of birth

Entity Relationship (ER) model

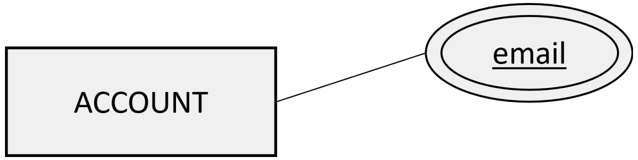
Attribute types: final remarks



- This is permissable: both the year of birth and name (incl. first and last name) make a person uniquely identifiable (in this model)

Entity Relationship (ER) model

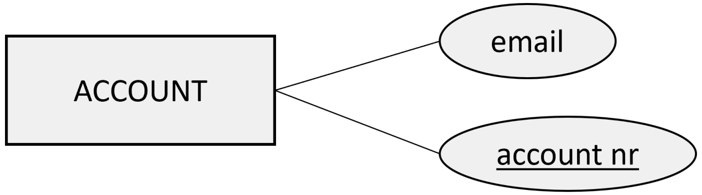
Attribute types: final remarks



- This too is ill-defined, i.e. is the combination of email adresses which need to be unique, or can every single email only occur once?
    - Better to redesign the model in this case

Entity Relationship (ER) model

Attribute types: final remarks



- Adding an attribute type "number" just to have something that makes the entity type unique
    - Makes sense in cases where the business context is already used to working with a number (e.g. a student number, course number)
    - In other cases: try to find a (combination of) other attribute types instead
- Might result in annoyances when converting the conceptual model to logical one (see later)
    - But remember: does not have to be a direct mapping, you are allowed to make modifications when translating to the logical model

Entity Relationship (ER) model

Relationship types

- Definition
- Degree and roles
- Cardinalities
- Relationship attribute types

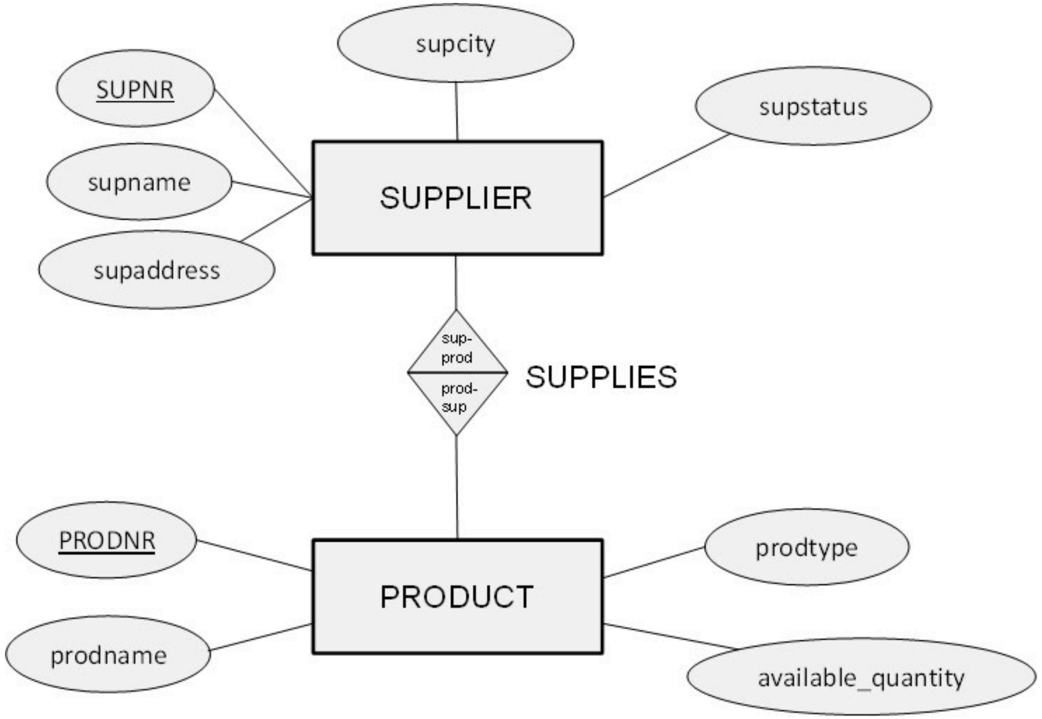
Entity Relationship (ER) model

Relationship 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
- In the ER model, relationship types are indicated using a rhombus symbol
  - Basically, the rhombus can be thought of as two adjacent arrows pointing to each of the entity types specifying both directions in which the relationship type can be interpreted
- A **relationship** is then a single instance of a relationship type

Entity Relationship (ER) model

Relationship types: definition



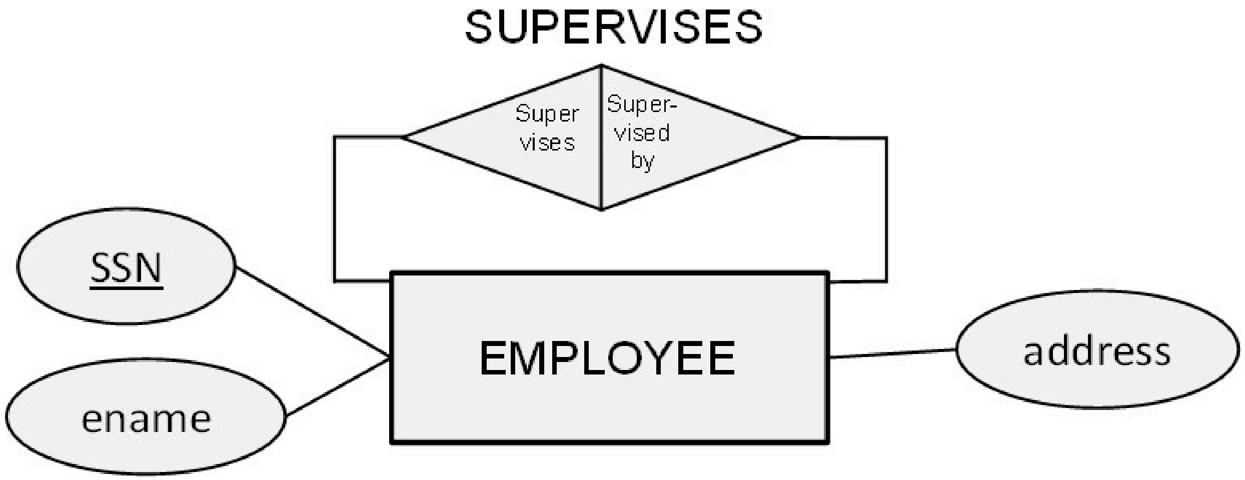
Entity Relationship (ER) model

Relationship types: degree and roles

- The **degree** of a relationship type corresponds to the number of *distinct* entity types participating in the relationship type
  - Not necessarily the same as the number of connections!
  - 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
  - Not always easy to write meaningful roles inside the rhombus and can optionally be left out

Entity Relationship (ER) model

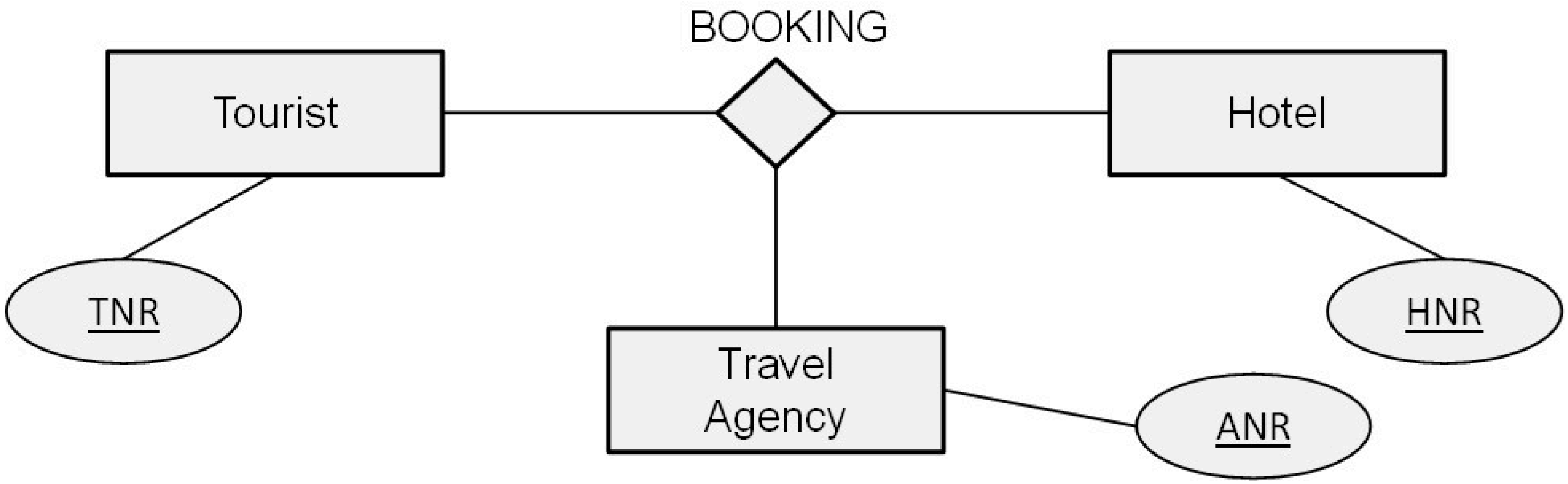
Relationship types: degree and roles



Unary degree (“recursive” relation)

Entity Relationship (ER) model

Relationship types: degree and roles



Ternary degree

Entity Relationship (ER) model

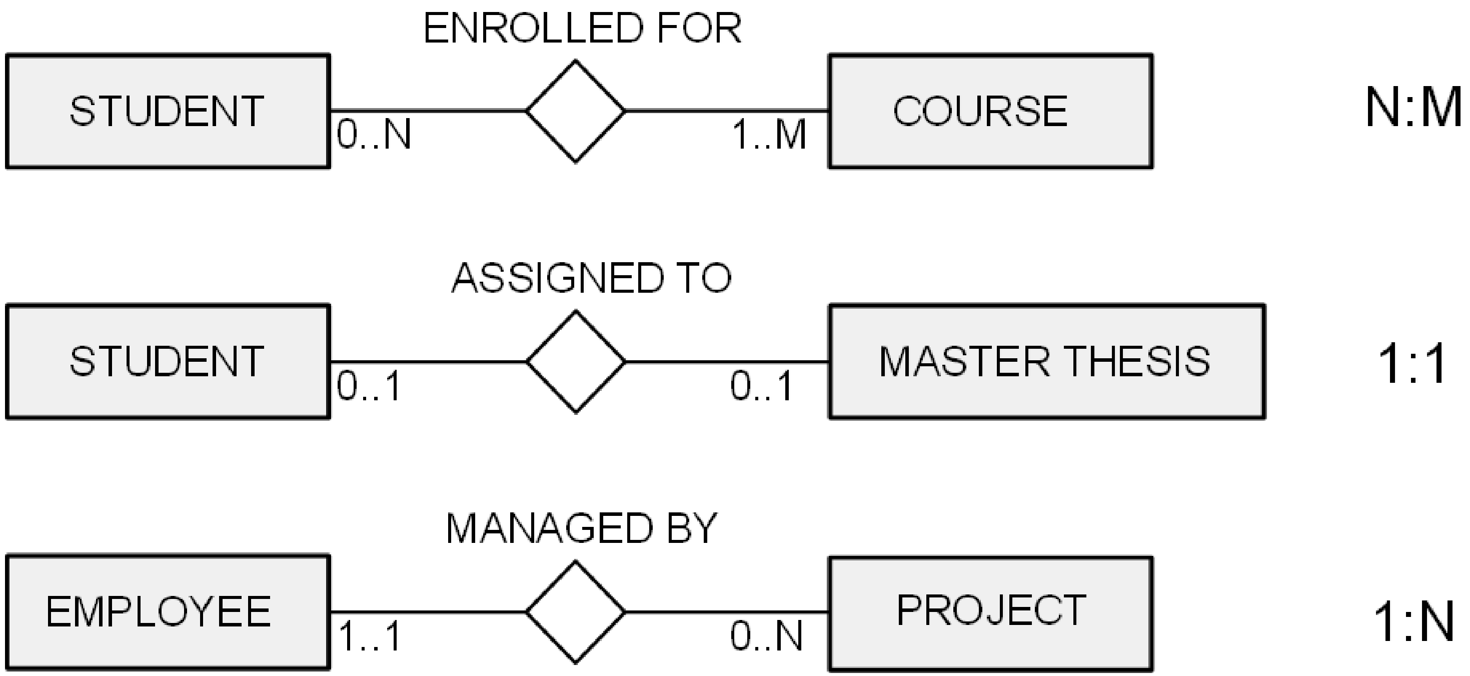
Relationship types: cardinalities

- 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 should/can participate in, given other entities participating in the relationship
- 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

Note: a minimum cardinality of 1 is very strict! Especially rare for ternary relationship types!

Entity Relationship (ER) model

Relationship types: cardinalities



ENROLLED FOR

STUDENT 0..N — 1..M COURSE

N:M

ASSIGNED TO

STUDENT 0..1 — 0..1 MASTER THESIS

1:1

MANAGED BY

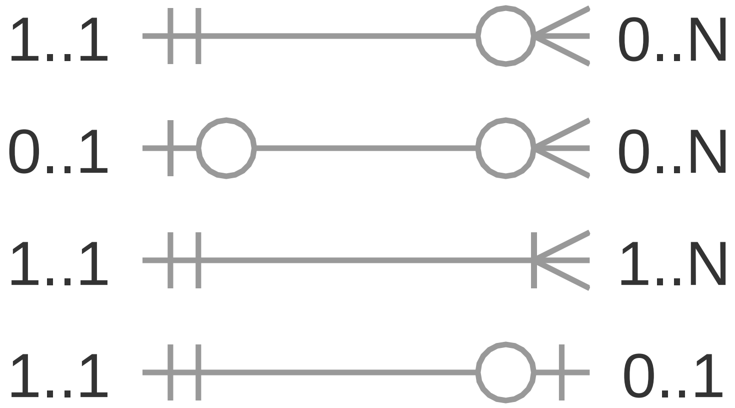
EMPLOYEE 1..1 — 0..N PROJECT

1:N

Entity Relationship (ER) model

Relationship types: cardinalities

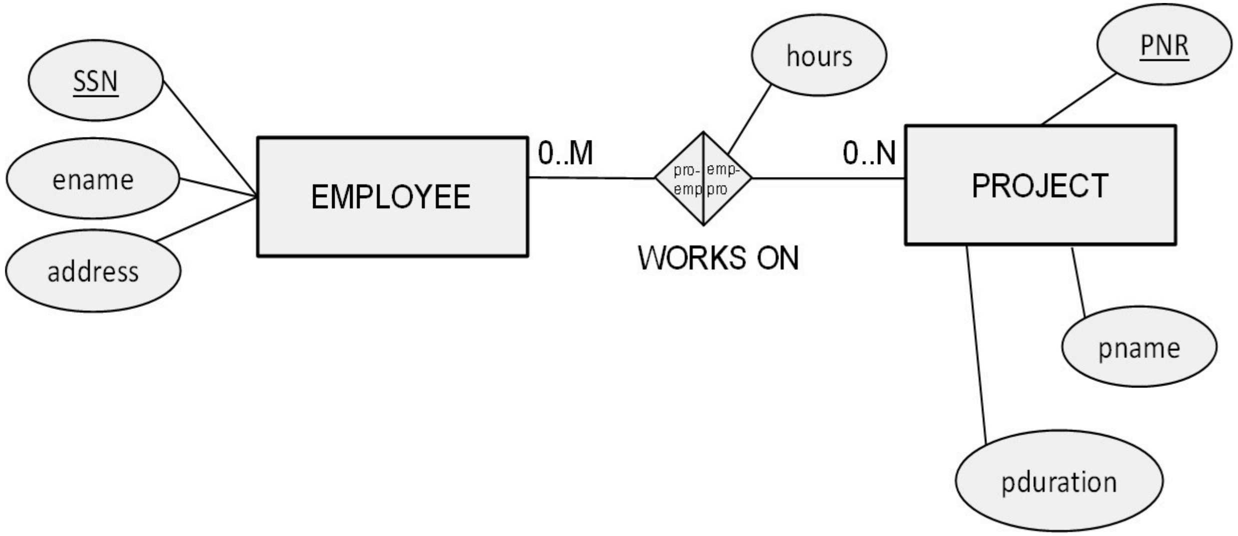
“Crow’s feet” notation also commonly used:



Entity Relationship (ER) model

Relationship types: relationship attribute types

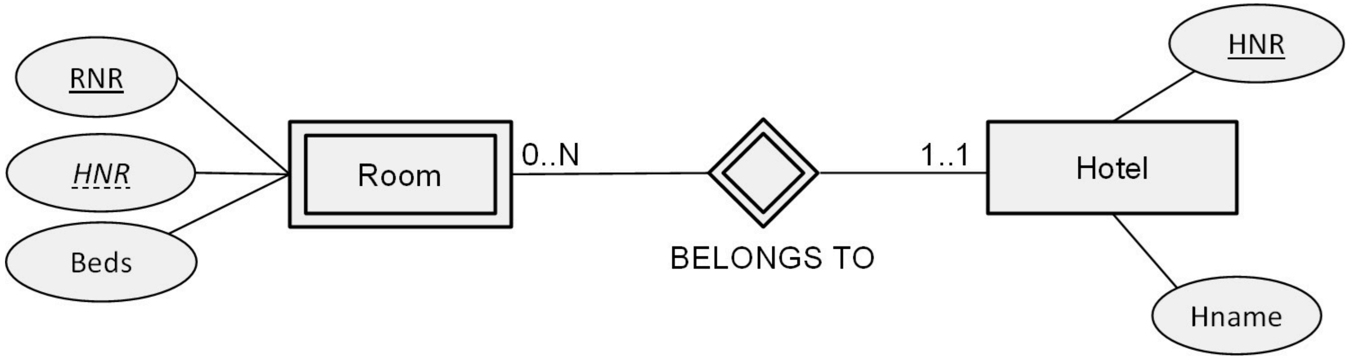
- Relationship types can also have attribute types
- These attribute types can be migrated to one of the participating entity types in case of a 1:1 or 1:N relationship type
- **No key attribute type** as the relation is uniquely defined by participating entities



Entity Relationship (ER) model

Weak entity types

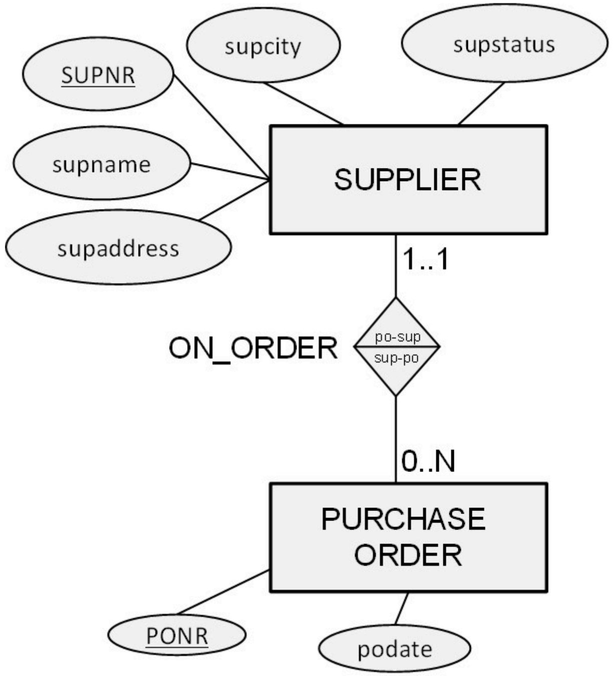
- A **strong entity type** is an entity type that has a key attribute type
- A **weak entity type** is an entity type that does not have a key attribute type of its own
  - And is hence related to owner entity type from which it borrows an attribute type to make up a key attribute type
- Weak entity types are represented in the ER model using a double bordered rectangle. The rhombus representing the relationship type through which the weak entity type borrows a key attribute type is double bordered as well. The borrowed attribute type(s) are underlined using a dashed line



Entity Relationship (ER) model

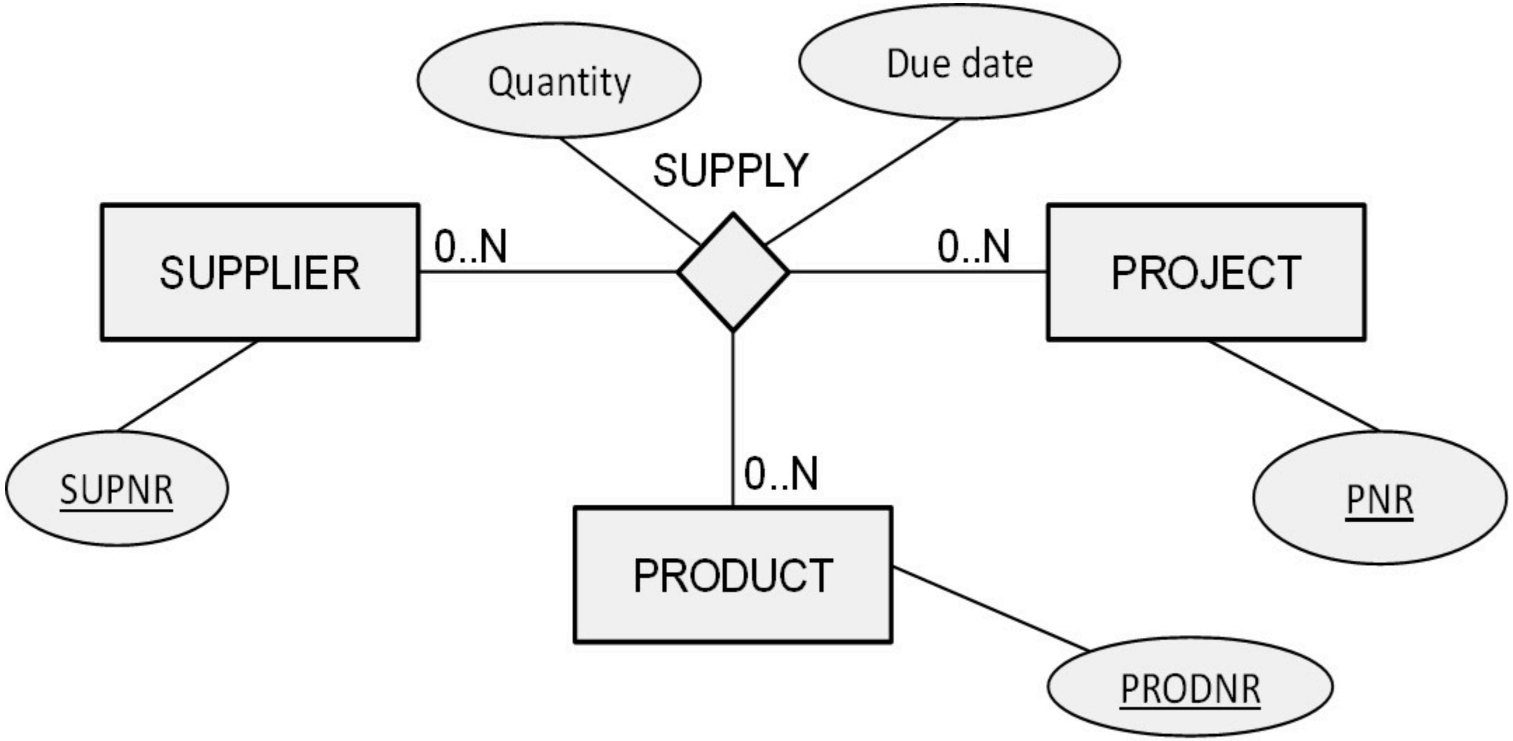
Weak entity types

- A weak entity type is always existence dependent from owner entity type (not vice versa: *existence dependency does not imply weak entity type*)



Entity Relationship (ER) model

Ternary relationship types



Entity Relationship (ER) model

Ternary relationship types

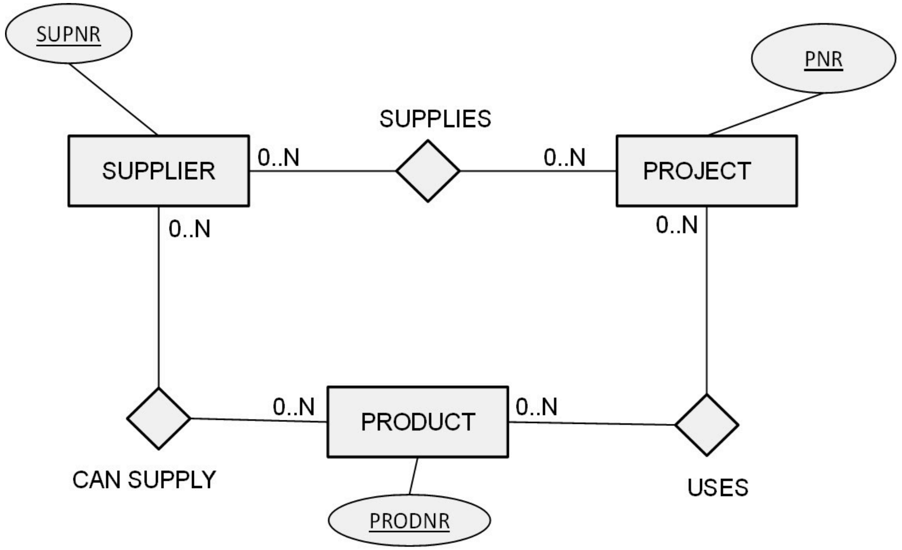
- The majority of relationship types in an ER model are binary or have only two participating entity types
- However, higher-order relationship types with more than two entity types, known as ternary relationship types, can occasionally occur, and special attention is needed to properly understand their meaning
  - Occasionally being the key word here! Don’t use constructs just “to show you used them”

Example:

- A particular supplier can supply a particular product for 0 to N projects
- A particular product for a particular project can be supplied by 0 to N suppliers
- A particular supplier can supply 0 to N products for a particular project
- The relationship type also includes the quantity and due date attribute types

Entity Relationship (ER) model

Ternary relationship types

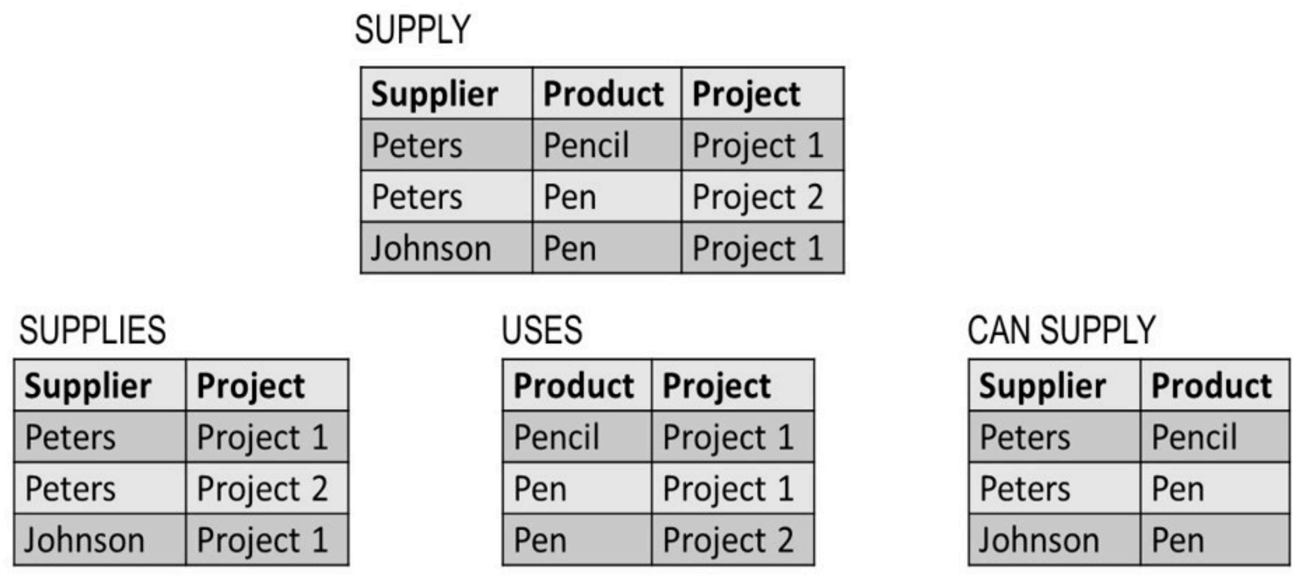


Loss of semantics!

Entity Relationship (ER) model

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

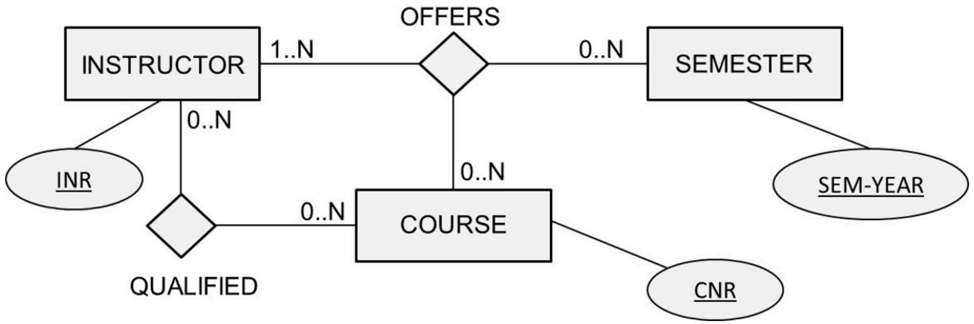
CAN SUPPLY

Supplier	Product
Peters	Pencil
Peters	Pen
Johnson	Pen

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

Entity Relationship (ER) model

Ternary relationship types



Binary relationship types can however be used to model *additional* semantics:

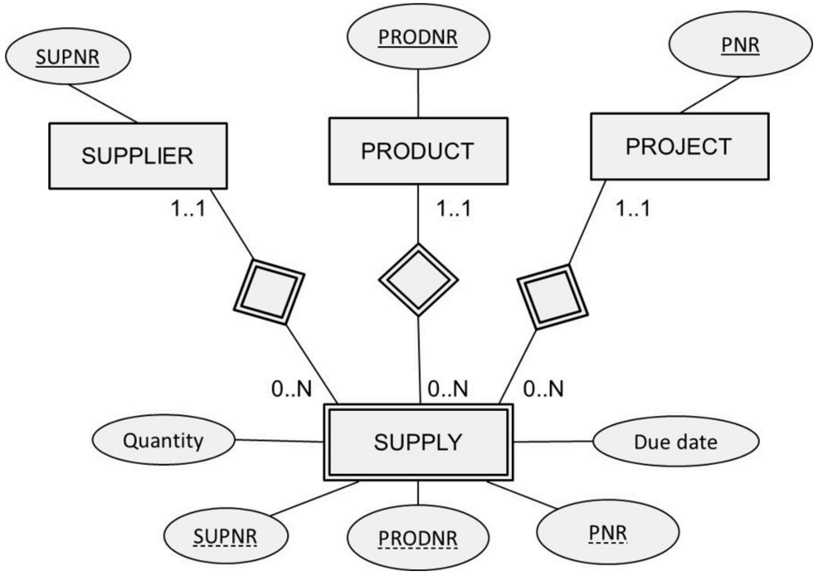
- An instructor can offer a course during 0 to N semesters. A course during a semester is offered by 1 to N instructors. An instructor can offer 0 to N courses during a semester
- In this case, we also added an extra binary relationship type QUALIFIED between INSTRUCTOR and COURSE to indicate what courses an instructor is qualified to teach
    - Note that it is possible that an instructor may be qualified for more courses than the ones they are teaching at the moment (and vice versa might not be qualified for a course they're teaching – the ER model alone does not enforce such constraints)

Entity Relationship (ER) model

Ternary relationship types

Another alternative to model a ternary relationship type is by using a weak entity type:

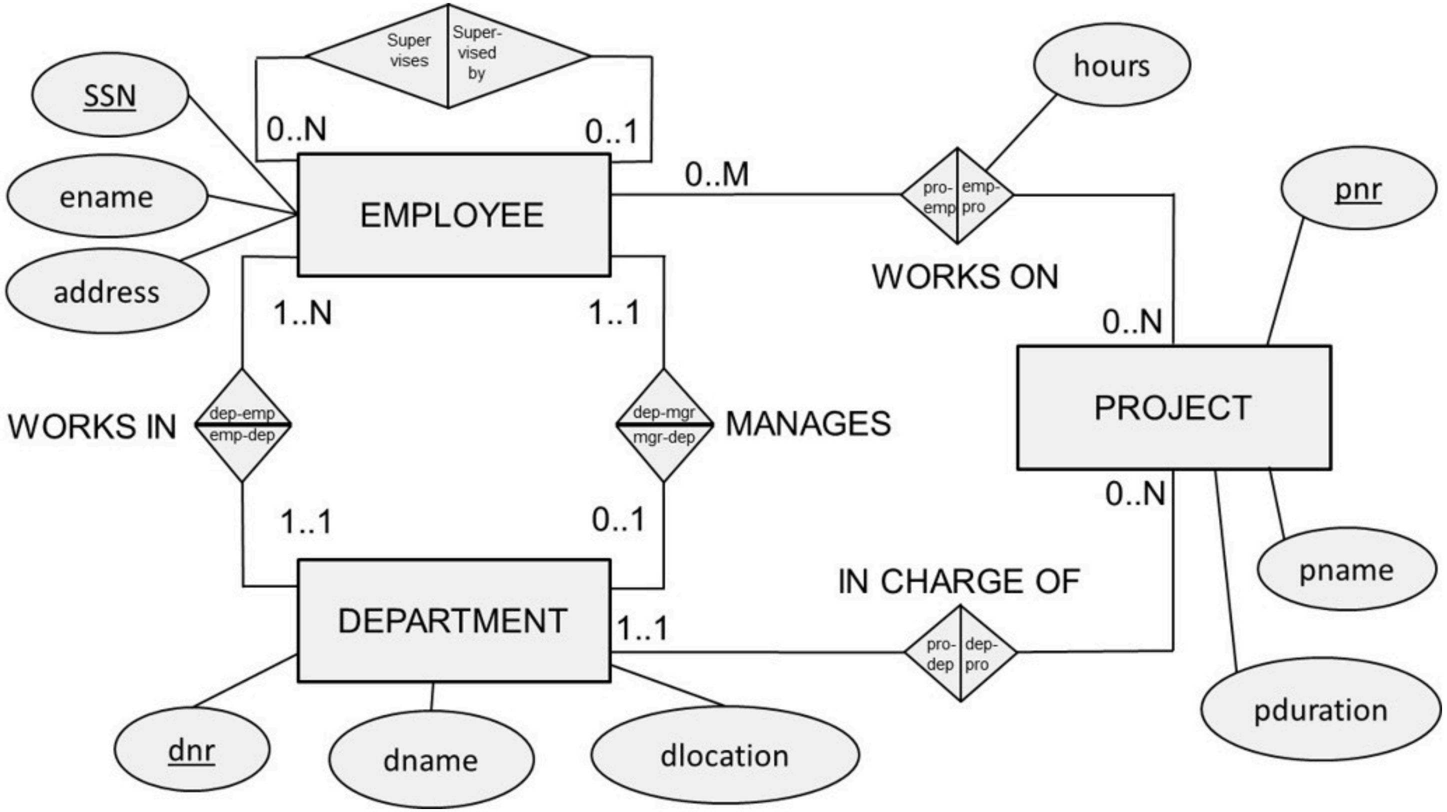
- The weak entity type SUPPLY is existence dependent on SUPPLIER, PRODUCT and PROJECT as indicated by the minimum cardinalities of 1. Its key is a combination of supplier number, product number and project number
- It also includes the attribute types quantity and due date



- Representing a ternary relationship type in this way can be handy in case the database modeling tool only supports unary and binary relationship types
- This is less commonly used

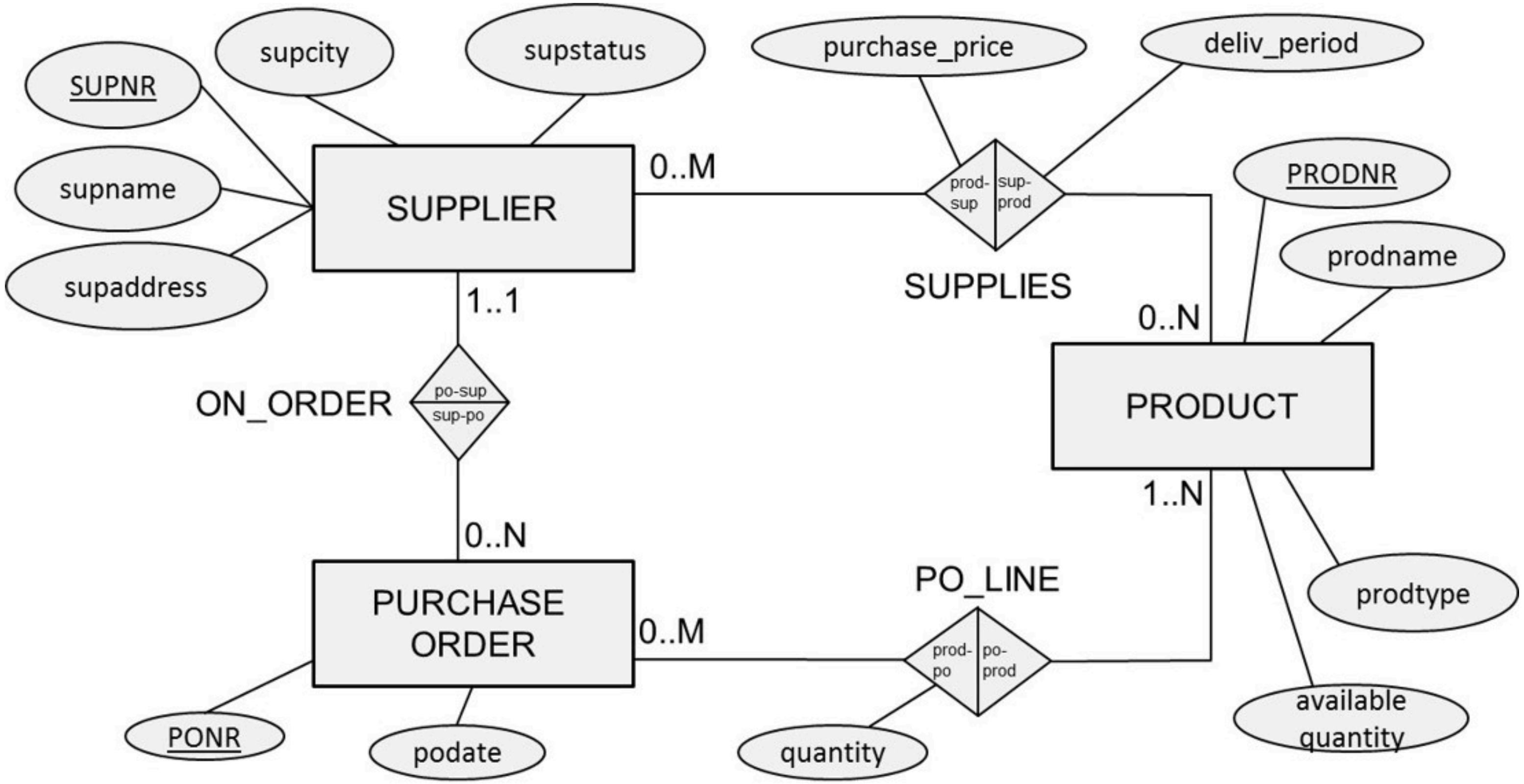
Entity Relationship (ER) model

Example 1



Entity Relationship (ER) model

Example 2



Entity Relationship (ER) model

Limitations

- 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.
  - Does not mean that “history” cannot be modeled by an ER model (history can be part of the “state”)
- ER model cannot guarantee **consistency across multiple relationship types**
  - Examples: an employee should work in the department that they manage, employees should work on projects assigned to departments to which they belong, suppliers can only be assigned to purchase orders for products they can supply
  - Put in documentation
- 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
  - Put in documentation
- 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

- The **enhanced entity relationship model or EER model** is an extension of the ER model
- It includes all the modelling concepts (entity types, attribute types, relationship types) of the ER model, as well as three new additional semantic data modelling concepts:
    - Specialization/generalization
    - Categorization
    - Aggregation

Note: none of these are strictly necessary in the sense that all of the situations that they cover can also be covered by the ER model (albeit with models looking more abstract)

Enhanced Entity Relationship (EER) model

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

Enhanced Entity Relationship (EER) model

Specialization/Generalization

Specialization refers to the process of defining a set of subclasses of an entity type

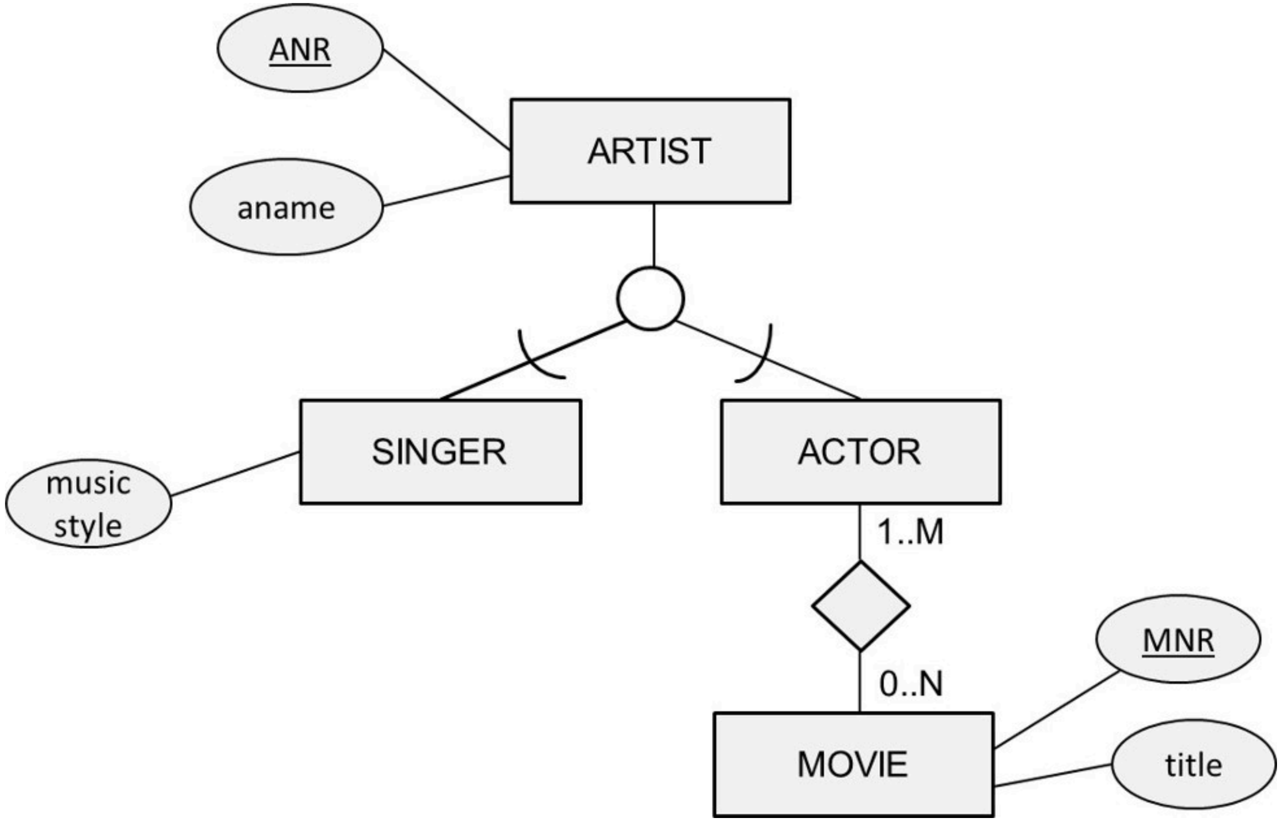
- Example: ARTIST superclass with subclasses SINGER and ACTOR
- The specialization defines an "IS A" relationship
- The specialization can then establish additional specific attribute types for each subclass
  - Example: singer can have a music style attribute type
- The specialization can also establish 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

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

Enhanced Entity Relationship (EER) model

Specialization/Generalization



Enhanced Entity Relationship (EER) model

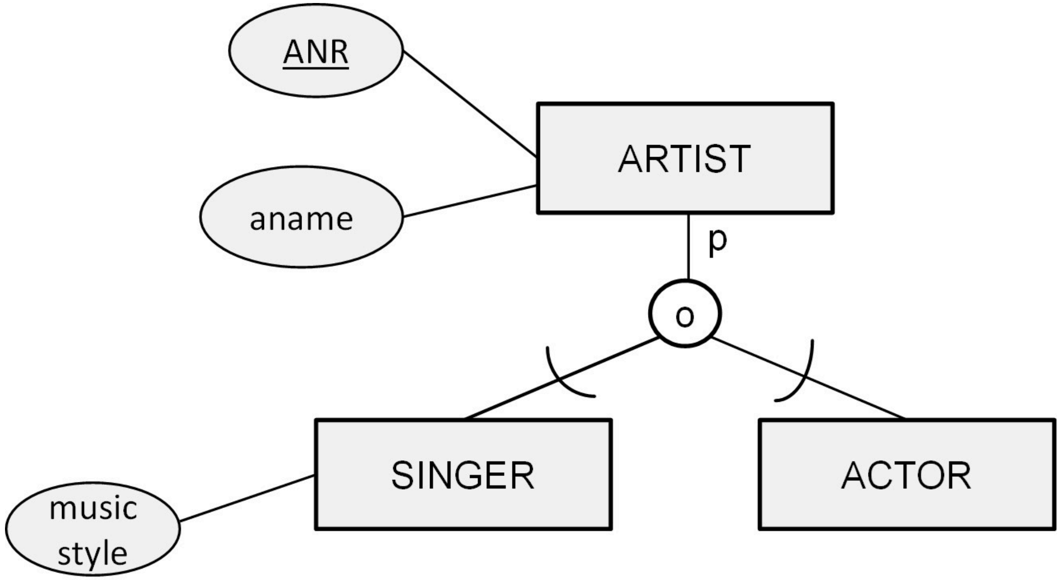
Specialization/Generalization: constraints

- 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 overlapping specialization is a specialization whereby the same entity may be a member of more than one subclass
  - Put inside a circle in the EER model (‘d’ or ‘o’)
- 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
  - Put next to the circle in EER model (‘t’ or ‘p’)

But: any entity belonging to a subclass also always belongs to its superclass!

Enhanced Entity Relationship (EER) model

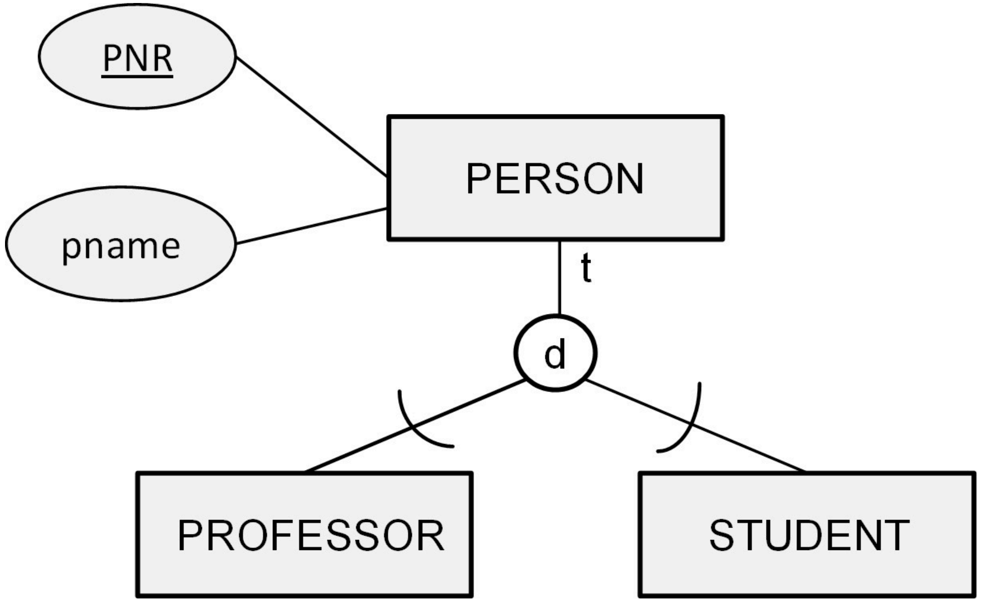
Specialization/Generalization: constraints



- The specialization is partial since not all artists are singers or actors (think about painters for example which are not included in our EER model)
- The specialization is overlapping since some artists can be both singers and actors

Enhanced Entity Relationship (EER) model

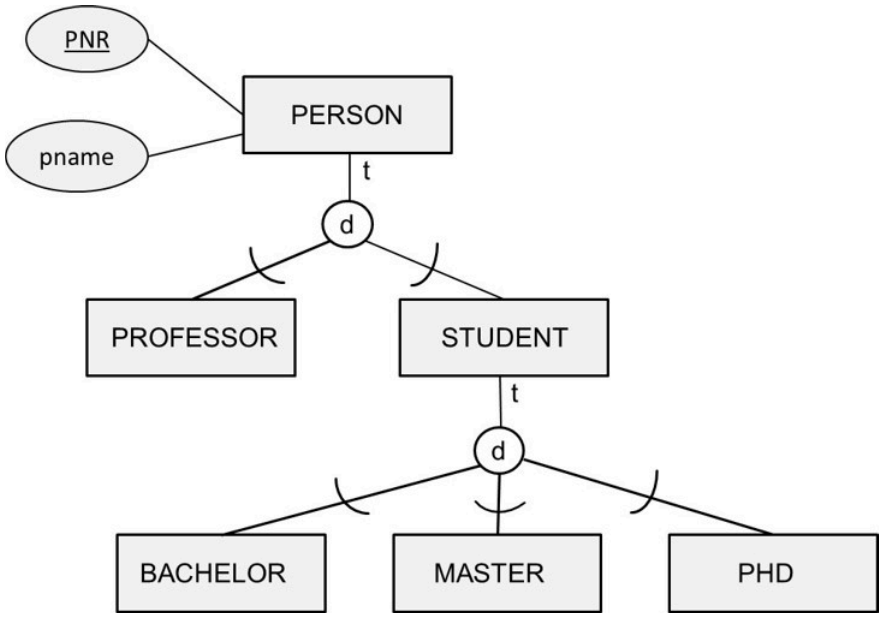
Specialization/Generalization: constraints



- The specialization is total, since according to our model, all people are either students or professors
- The specialization is disjoint, since a student cannot be a professor at the same time

Enhanced Entity Relationship (EER) model

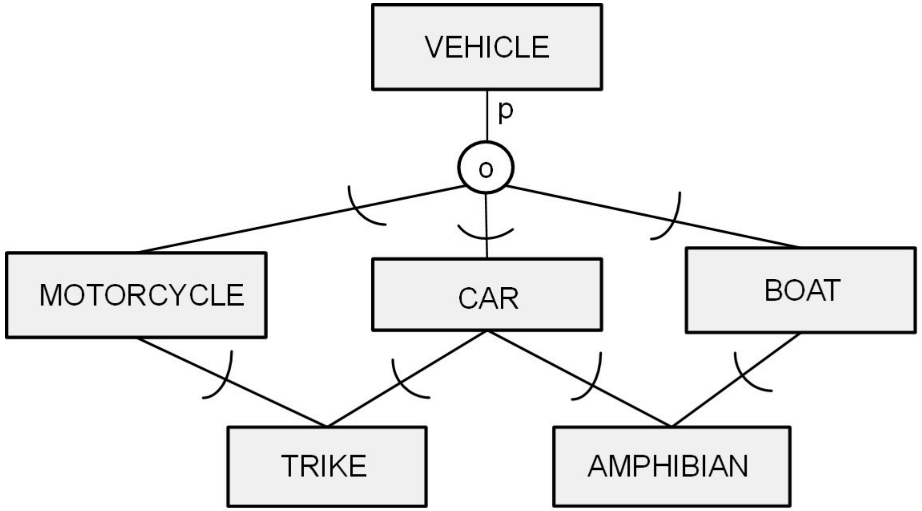
Specialization/Generalization: hierarchies



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

Enhanced Entity Relationship (EER) model

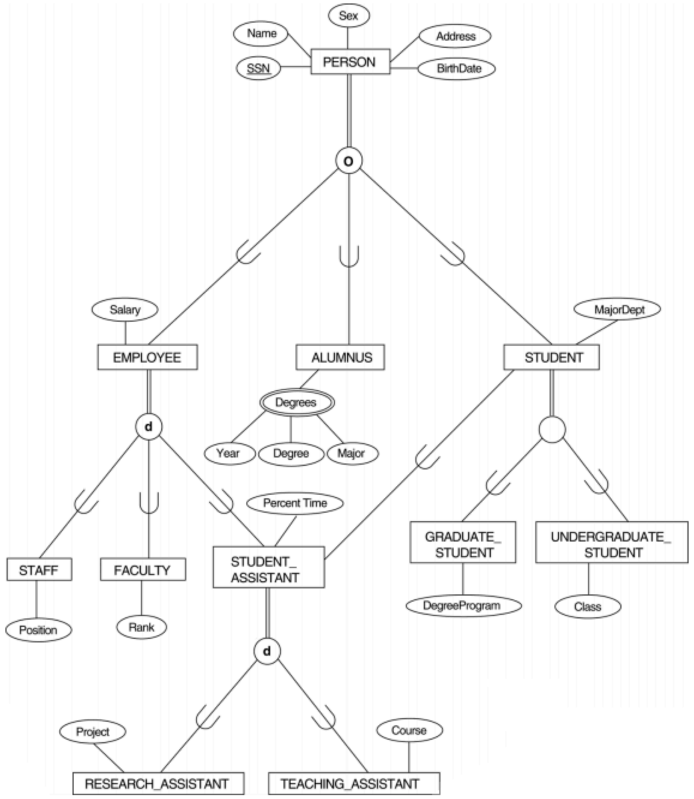
Specialization/Generalization: lattices



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

Enhanced Entity Relationship (EER) model

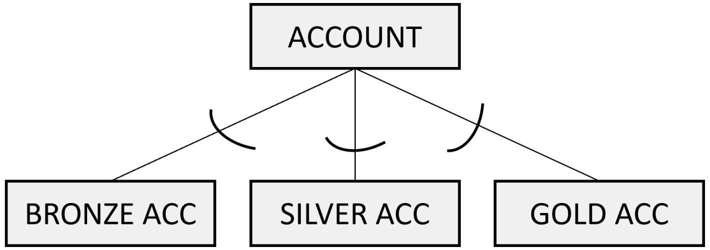
Multiple inheritance example



Enhanced Entity Relationship (EER) model

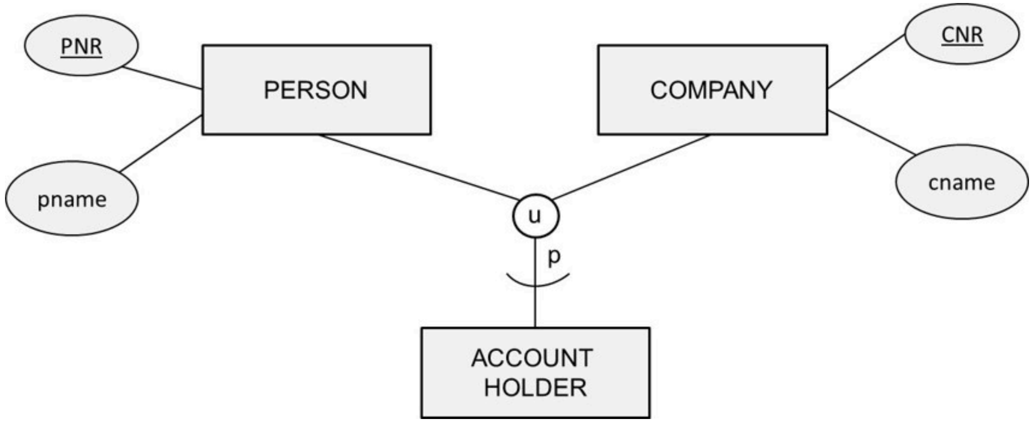
Specialization/Generalization: lattices and multiple inheritance

- Don't go overboard and design huge specialization trees!
  - Also not in OO programming languages
- Reminder: every construct of EER can also be modeled by an ER model
- Also: entities belong to their same type during their life cycle
  - This is perhaps not the best idea (abstraction):



Enhanced Entity Relationship (EER) model

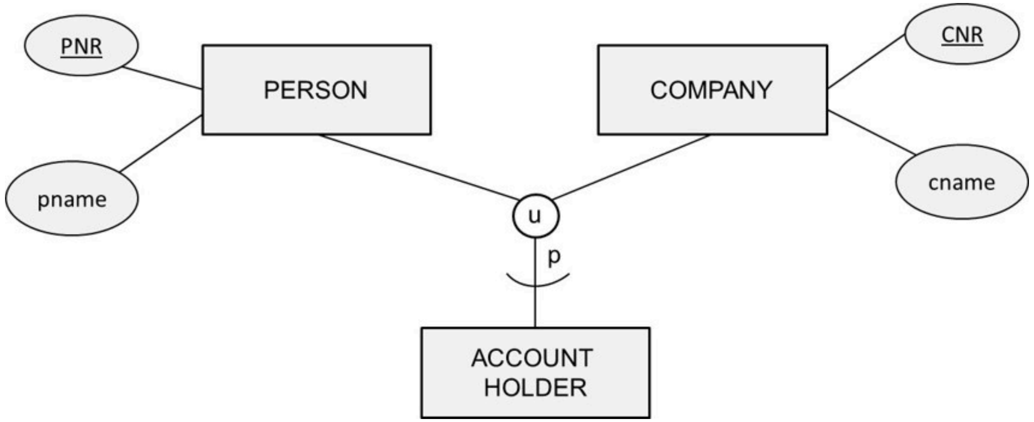
Categorization



- A **category** (or union type) 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
- A categorization is represented in the EER model by a circle with a letter ‘u’ (from union) in it

Enhanced Entity Relationship (EER) model

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

Enhanced Entity Relationship (EER) model

Categorization

Selective inheritance means that some ACCOUNT HOLDERs can be PERSONs, some ACCOUNT HOLDERs are COMPANY(ies), and some ACCOUNT HOLDERs are both

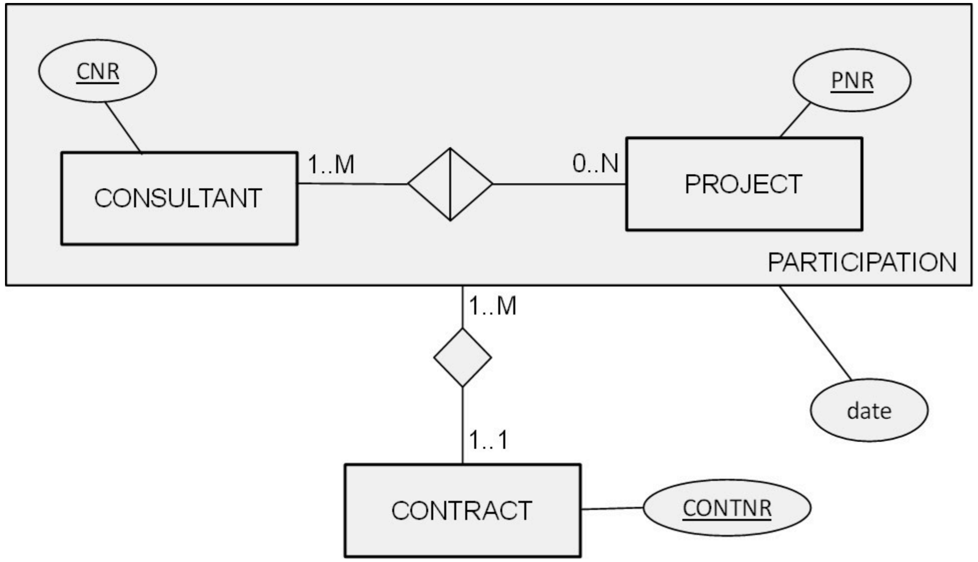
- Total or partial indicates whether all PERSON and COMPANY classes should belong to the subclass or not

Here, too it’s often better to simply model this as a normal specialization/generalization instead or find another way to represent this

Enhanced Entity Relationship (EER) model

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 involvement relationship types
- Can also be modeled by introducing a new entity type
    - This is generally the solution when in doubt how to handle additional complexity in ER models: entity types being the most “expressive” building block



Enhanced Entity Relationship (EER) model

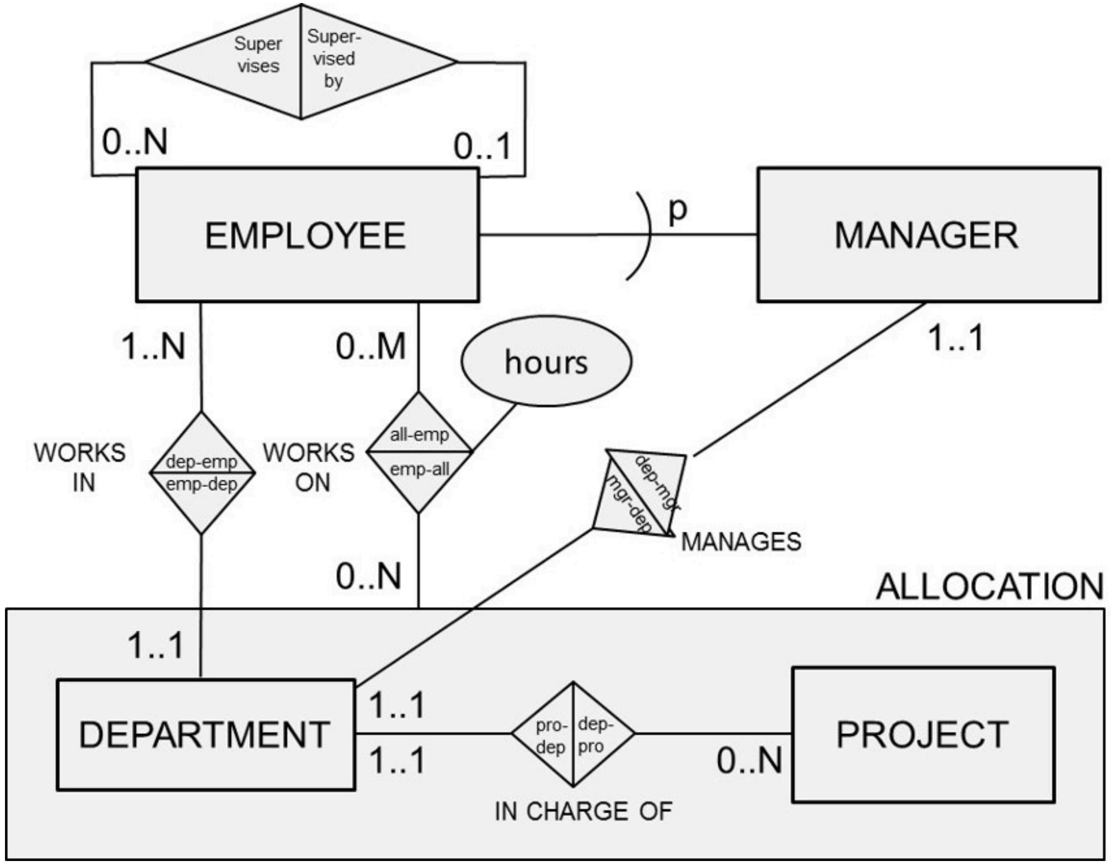
Aggregation

- A consultant works on 0 to N projects
- A project is being worked upon by 1 to M consultants
- Both entity types and the corresponding relationship type can now be aggregated into the aggregate concept PARTICIPATION
- This aggregate has its own attribute type date, which represents the date at which a consultant started working on a project
- The aggregate also participates in a relationship type with CONTRACT
- A participation should lead to minimum 1 and maximum 1 contract
- Conversely, a contract can be based upon 1 to M participations of consultants in projects

Aggregation here combines a single project and consultant!

Enhanced Entity Relationship (EER) model

Example



Enhanced Entity Relationship (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

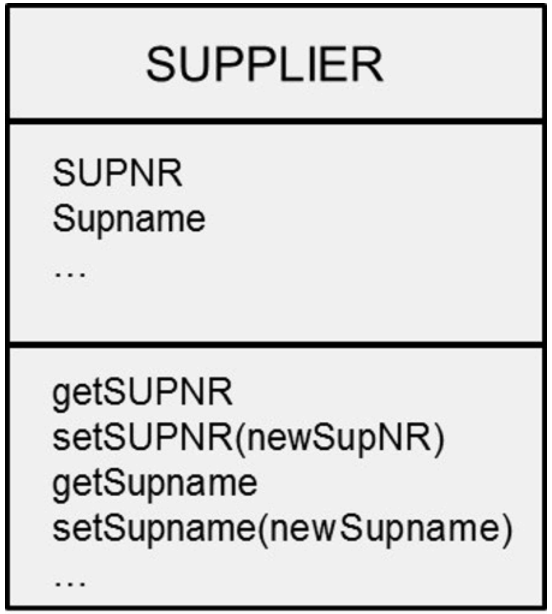
UML class diagram

- 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
- 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 is typically used to model object oriented applications — and is more commonly used in those settings. In what follows, we approach it as a conceptual tool!

UML class diagram

- In a UML class diagram, a **class** is represented as a rectangle with three sections
- In the upper part, the name of the class is mentioned, in the middle part the variables and in the bottom part the methods
- Methods seldomly used for conceptual data modelling



SUPPLIER

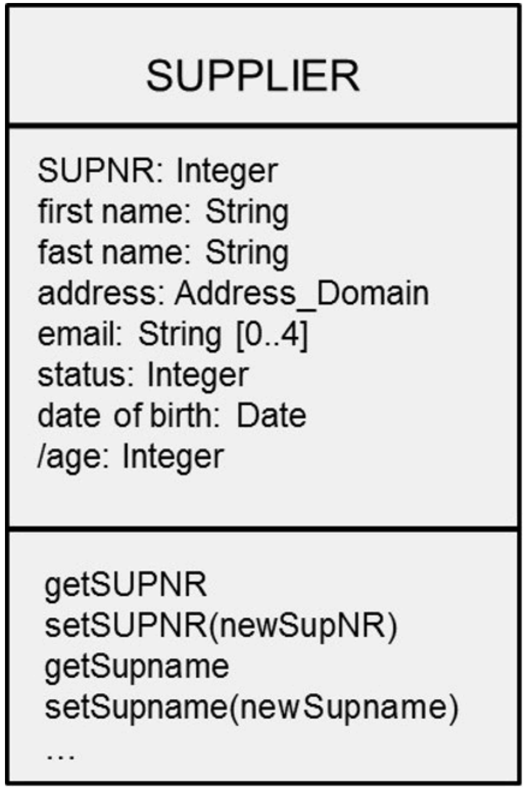
SUPNR
Supname
...

getSUPNR
setSUPNR(newSupNR)
getSupname
setSupname(newSupname)
...

UML class diagram

Variables

- Variables with unique values (comparable to key attribute types in ER) are not directly supported in UML
  - The reason is because a UML class diagram is assumed to be implemented using an object oriented environment (language, DBMS) in which every object created is assigned a unique and immutable object identifier (OID) which it keeps during its entire lifetime. Hence, this OID can be used to uniquely identify objects and no other variables are needed to serve as a key
  - To explicitly enforce the uniqueness constraint of a variable, you can however use OCL as we discuss in what follows
- 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 (‘[0..4]’) / Derived (‘/’) variables are possible



SUPPLIER

SUPNR: Integer
first name: String
fast name: String
address: Address_Domain
email: String [0..4]
status: Integer
date of birth: Date
/age: Integer

getSUPNR
setSUPNR(newSupNR)
getSupname
setSupname(newSupname)
...

UML class diagram

Associations

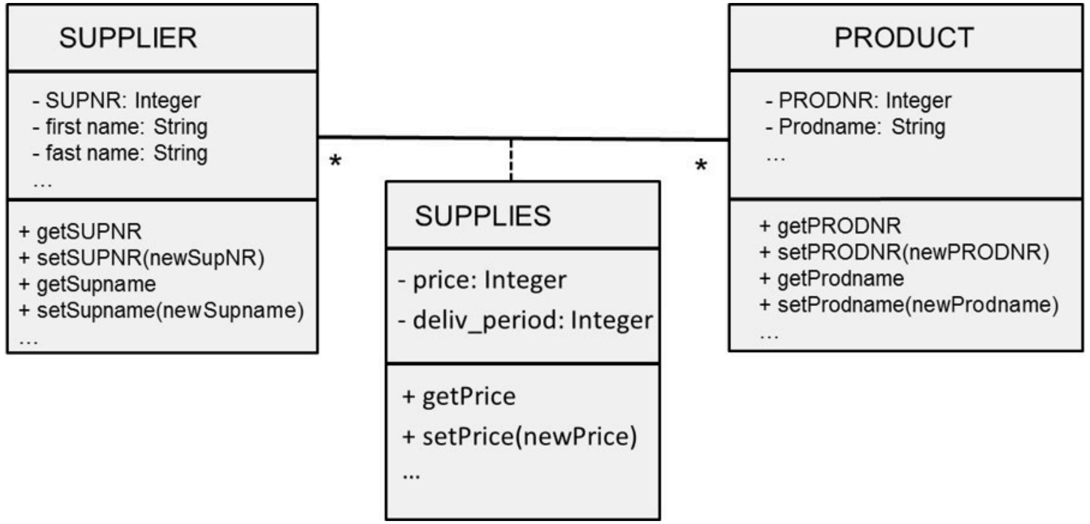
- An association corresponds to a relationship type in ER
- Multiple associations can be defined between the same classes
- A particular occurrence of an association is referred to as a link
- An association is characterized by its multiplicities (cardinalities in the ER model)

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

UML class diagram

Associations: association class

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



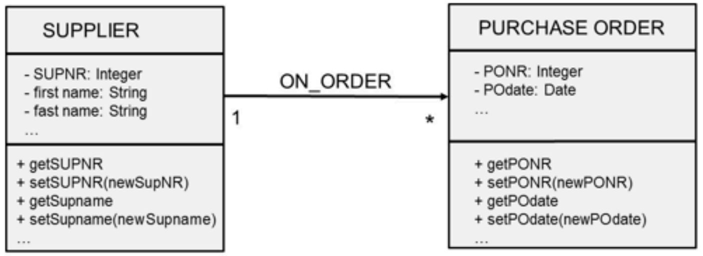
Otherwise: only the association connection itself is drawn

UML class diagram

Associations: unidirectional versus bidirectional association

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

Unidirectional



SUPPLIER

- SUPNR: Integer
- first name: String
- fast name: String
...

+ getSUPNR
+ setSUPNR(newSupNR)
+ getSupname
+ setSupname(newSupname)
...

ON_ORDER

1

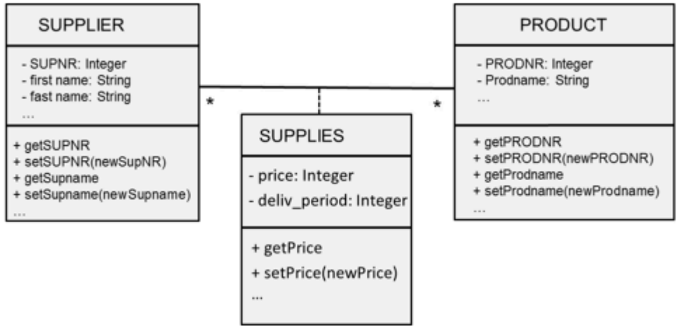
*

PURCHASE ORDER

- PONR: Integer
- POdate: Date
...

+ getPONR
+ setPONR(newPONR)
+ getPOdate
+ setPOdate(newPOdate)
...

Bidirectional



SUPPLIER

- SUPNR: Integer
- first name: String
- fast name: String
...

+ getSUPNR
+ setSUPNR(newSupNR)
+ getSupname
+ setSupname(newSupname)
...

*

*

SUPPLIES

- price: Integer
- deliv_period: Integer

+ getPrice
+ setPrice(newPrice)
...

PRODUCT

- PRODNR: Integer
- Prodname: String
...

+ getPRODNR
+ setPRODNR(newPRODNR)
+ getProdname
+ setProdname(newProdname)
...

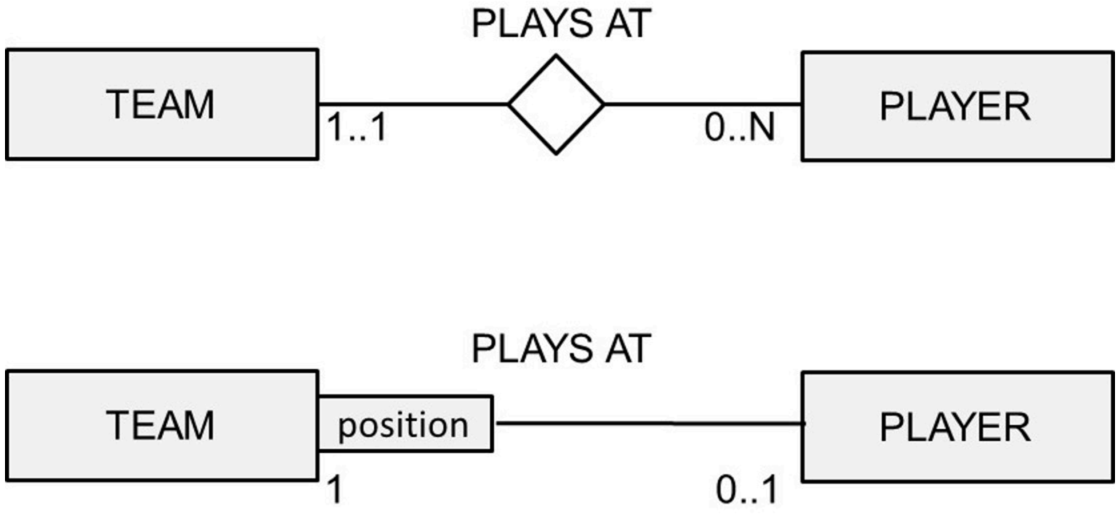
UML class diagram

Associations: 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

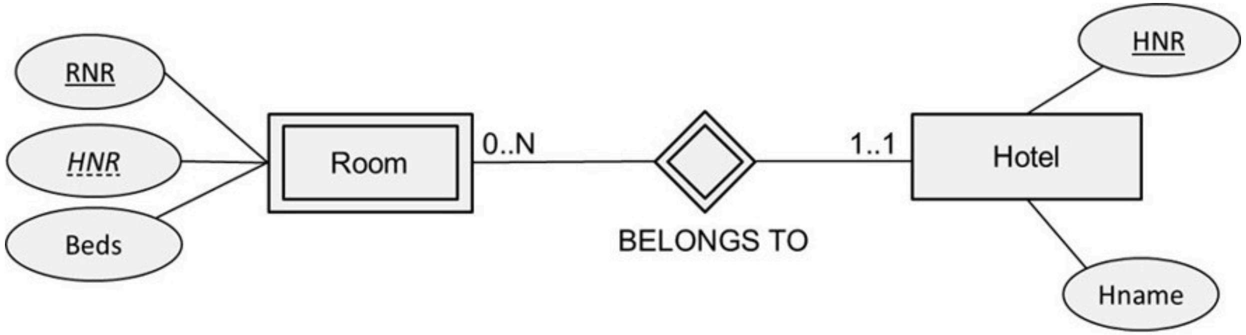
UML class diagram

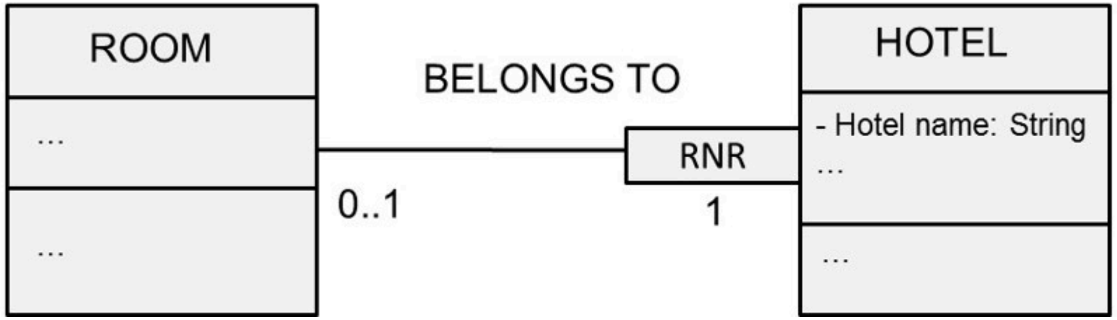
Associations: qualified association



UML class diagram

Associations: qualified association



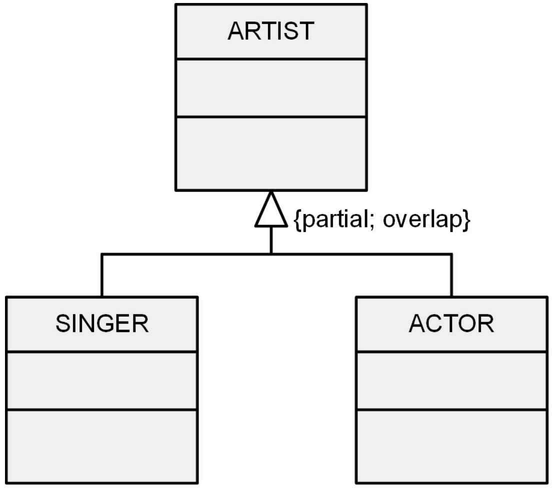


UML class diagram

Specialization/generalization

Similar to the EER model, UML also supports specialization or generalization relationships: hollow triangle represents a specialization in UML

- The specialization characteristics such as total/partial or disjoint/overlap can be added next to the triangle
- UML also supports multiple inheritance whereby a subclass can inherit variables, methods and associations from multiple superclasses



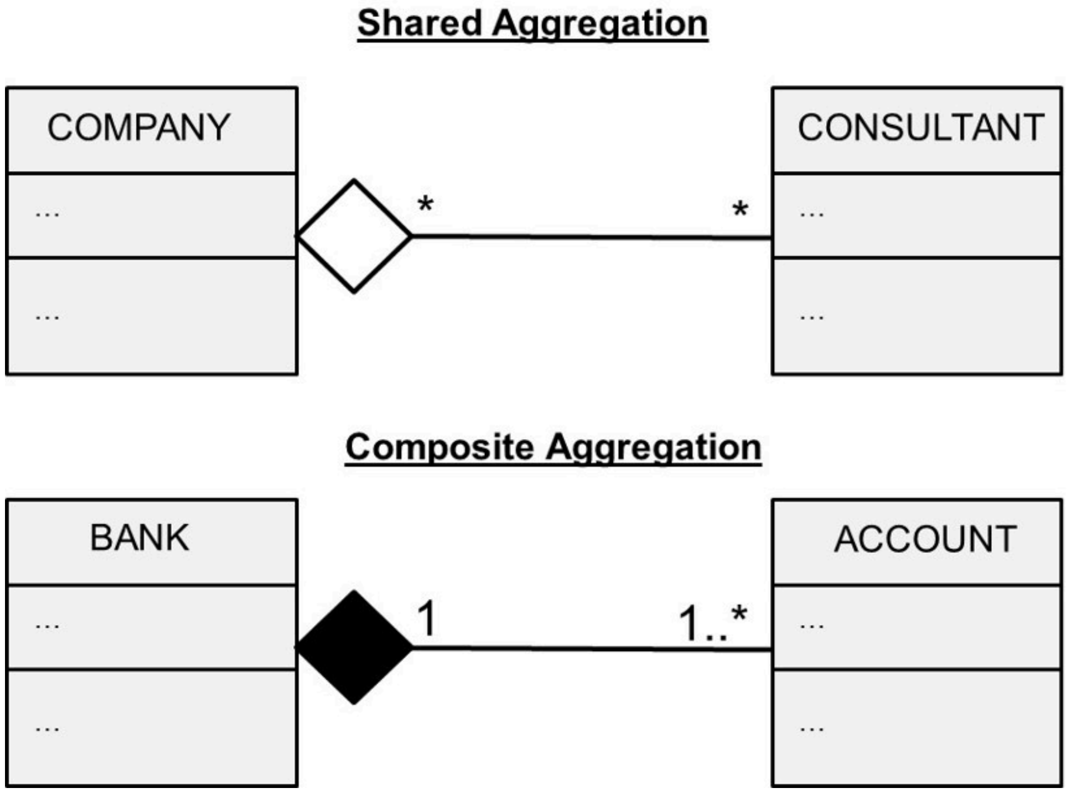
UML class diagram

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
- 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

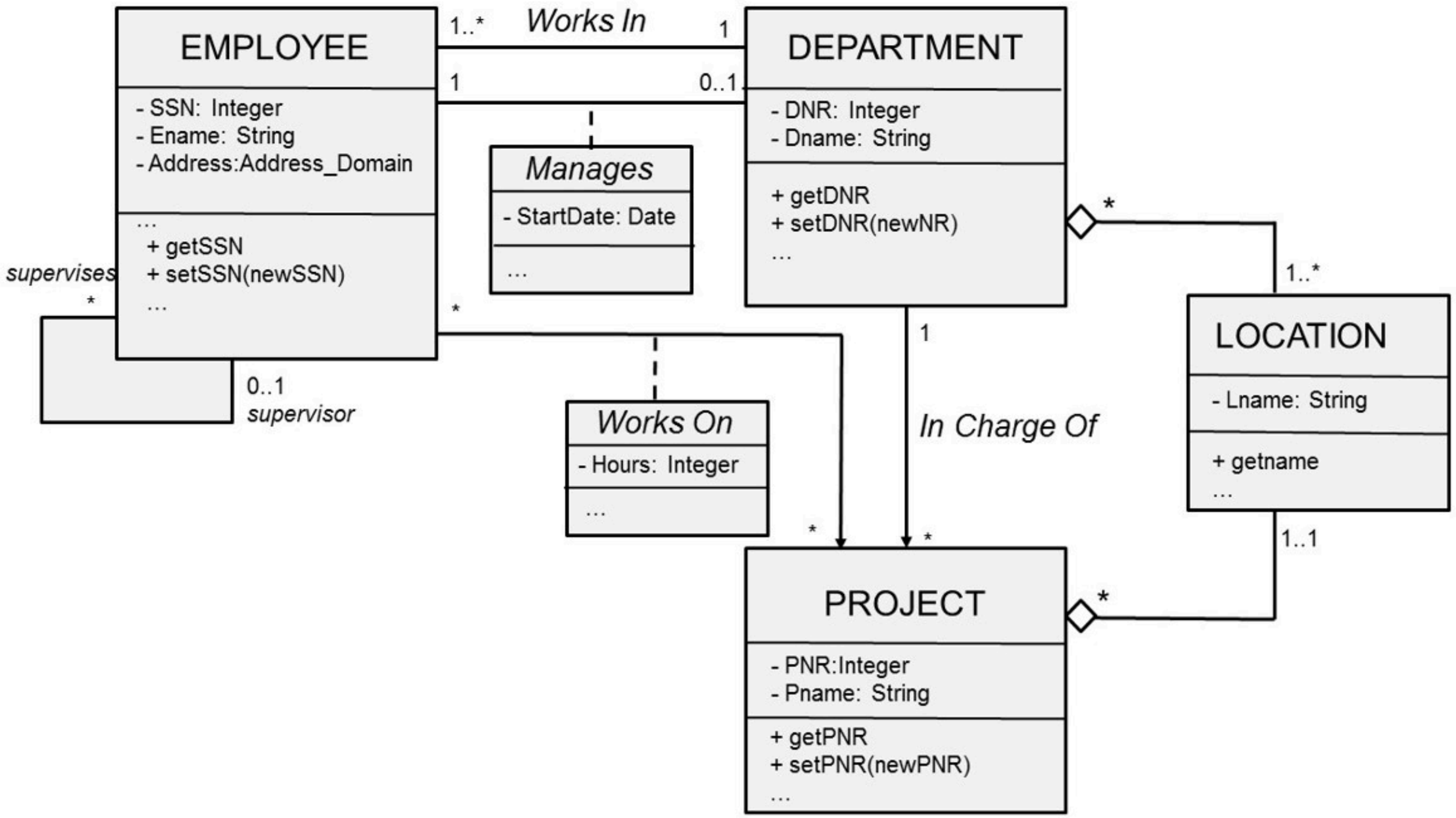
UML class diagram

Aggregation



UML class diagram

Example



UML class diagram

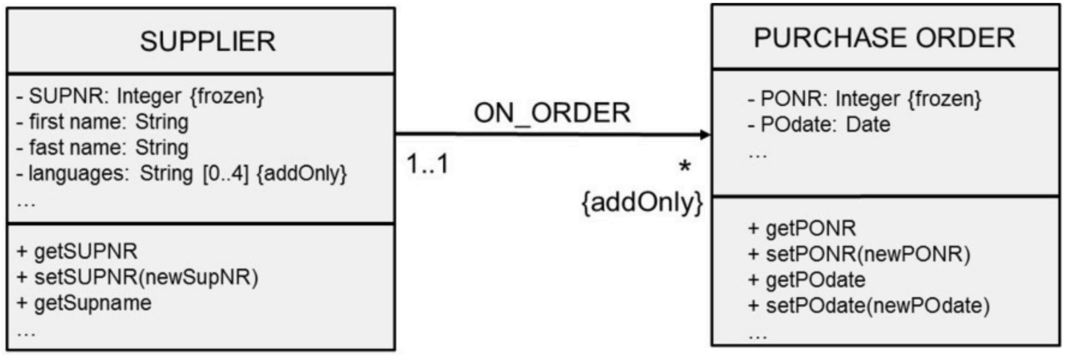
Advanced concepts

- Changeability property
- Object Constraint Language (OCL)

UML class diagram

Advanced concepts: 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
- This is a feature which would in fact also be helpful in ER models



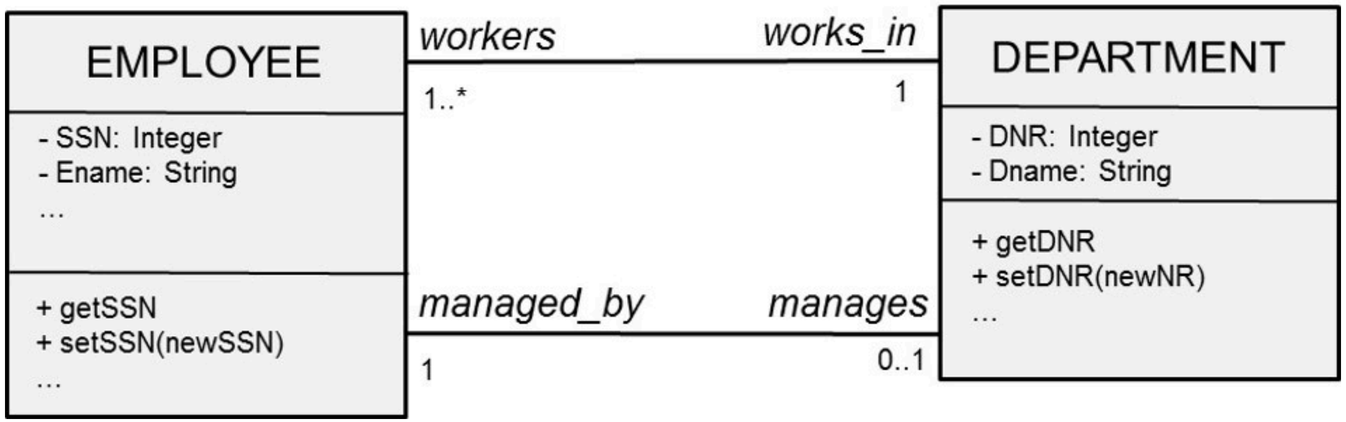
UML class diagram

Advanced concepts: 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/
- 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

UML class diagram

Advanced concepts: Object Constraint Language (OCL) example

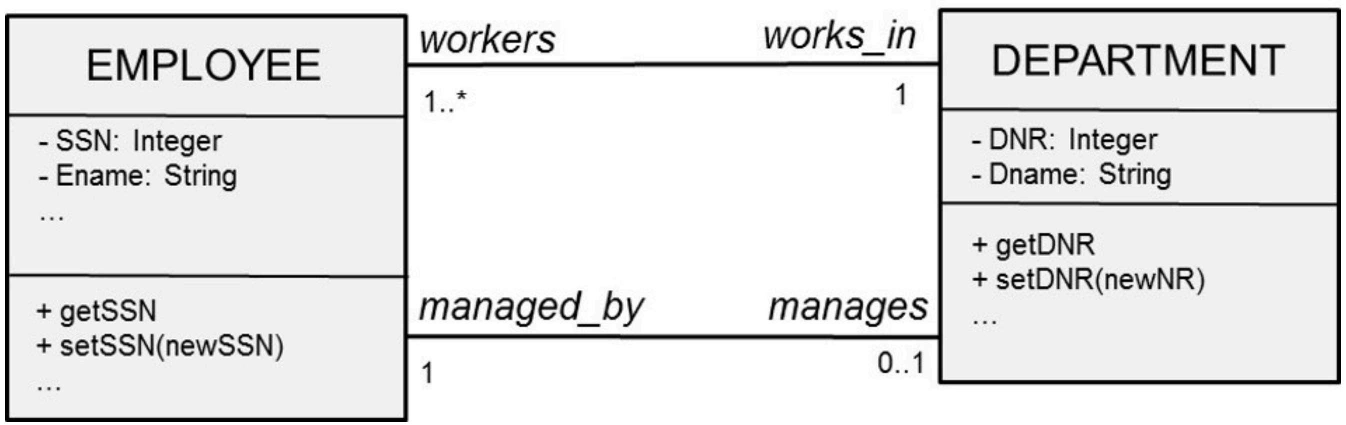


Constraint: manager of a department should be at least 10 years employed

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

UML class diagram

Advanced concepts: Object Constraint Language (OCL) example

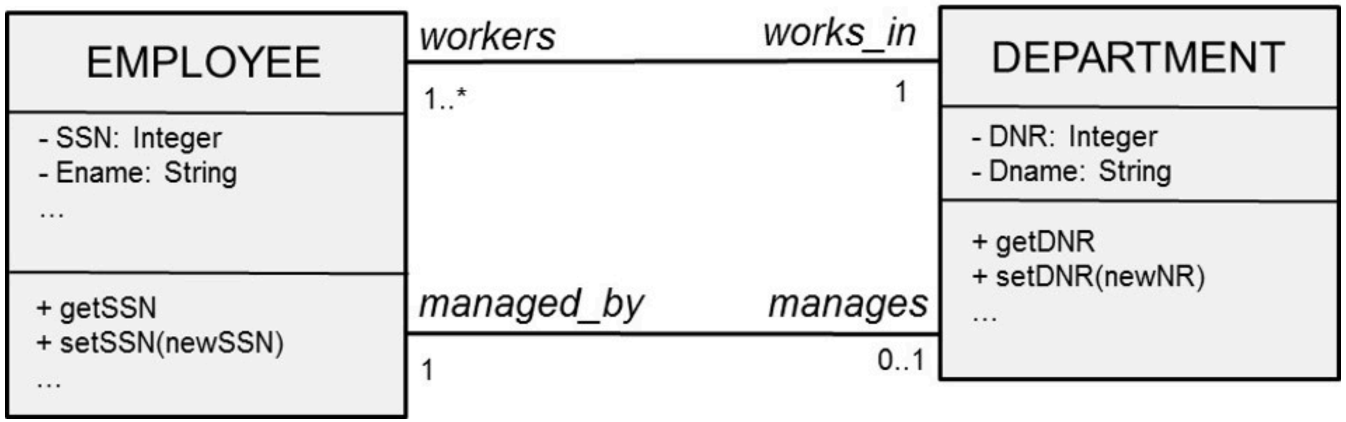


A department should have at least 20 employees

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

UML class diagram

Advanced concepts: Object Constraint Language (OCL) example



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

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

UML class diagram

UML class diagram versus EER

UML class diagram	EER model
Class	Entity type
Object	Entity
Variable	Attribute type
OID (~, or through OCL)	Key attribute type
Variable value	Attribute
Method	*NA*
Association	Relationship type
Link	Relationship
Multiplicity	Cardinality
Qualified association	Weak entity type (~)
Specialization/Generalization	Specialization/Generalization
Aggregation (composite, shared)	Aggregation
OCL	*NA*