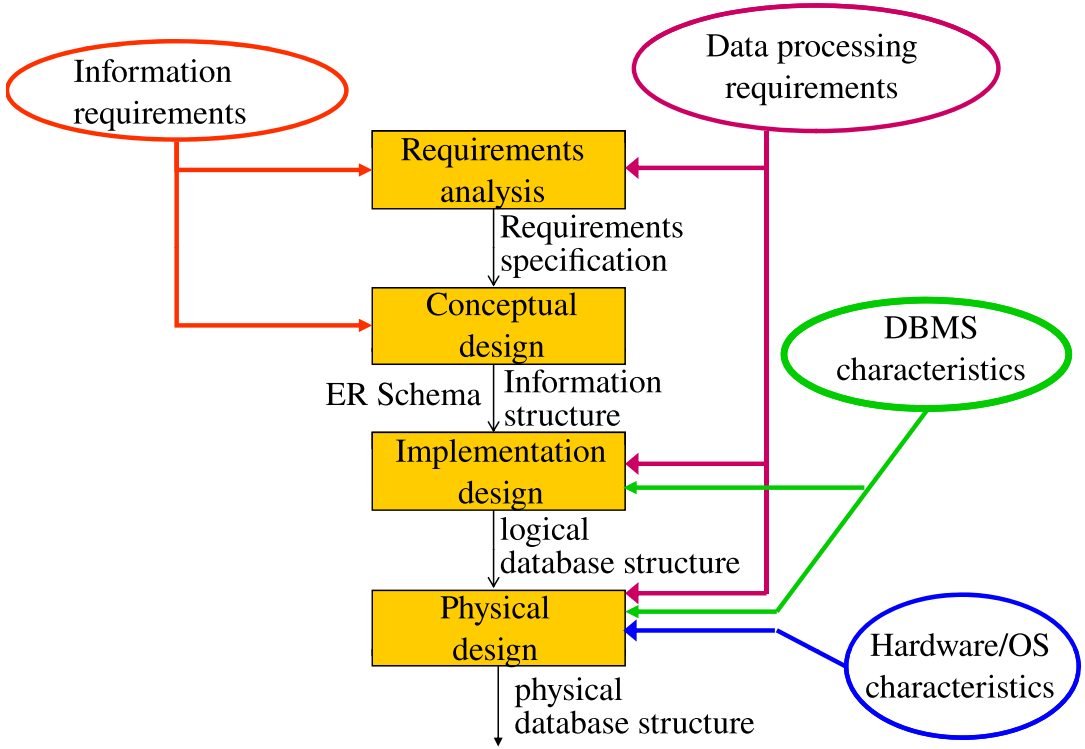
Database Design

Database Design

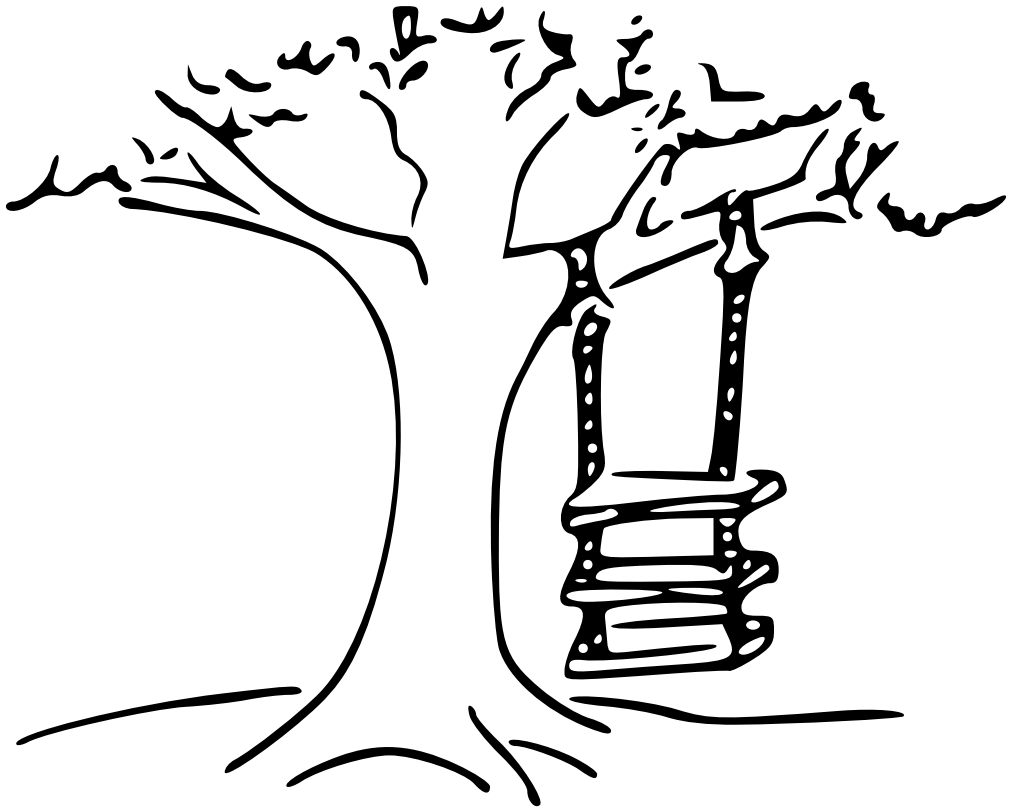
Phases of Database Design



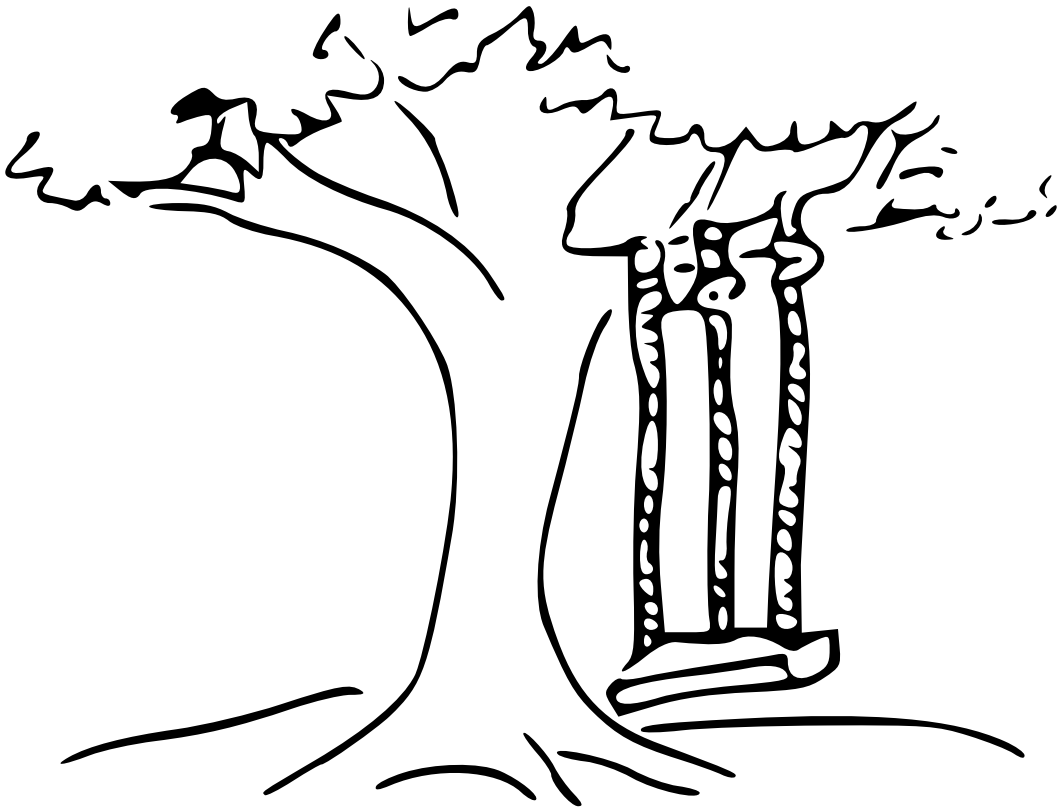
Requirements Analysis

- Discussed in more detail in software engineering lectures
- Here we only briefly touch on this area
- Very important point in projects: users and developers understand each other

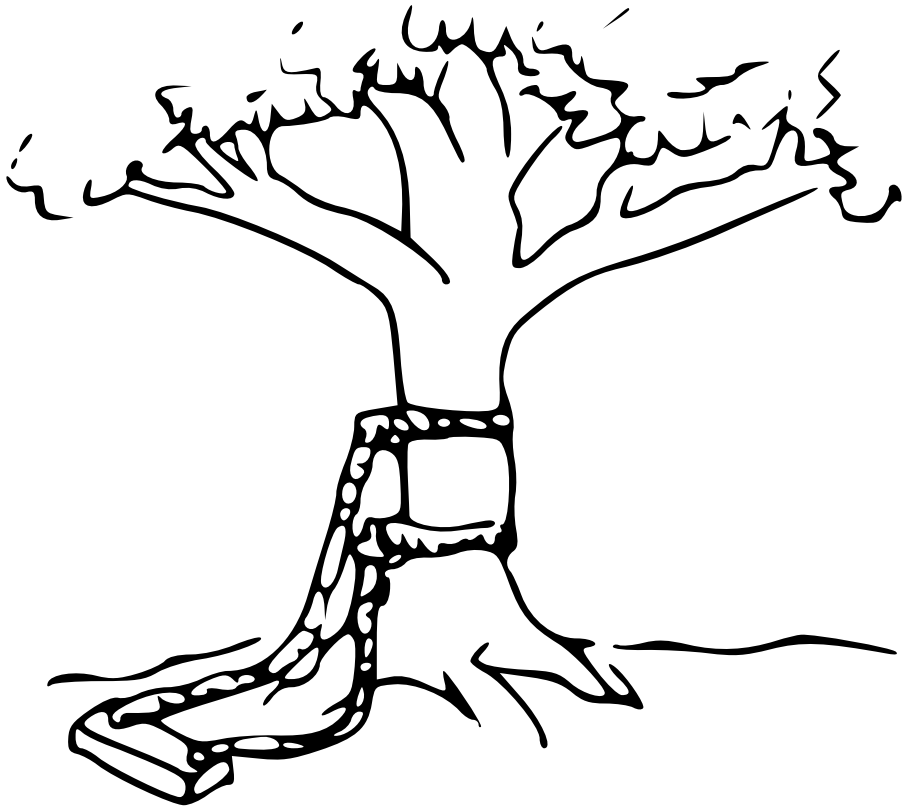
How the sponsor proposed it:



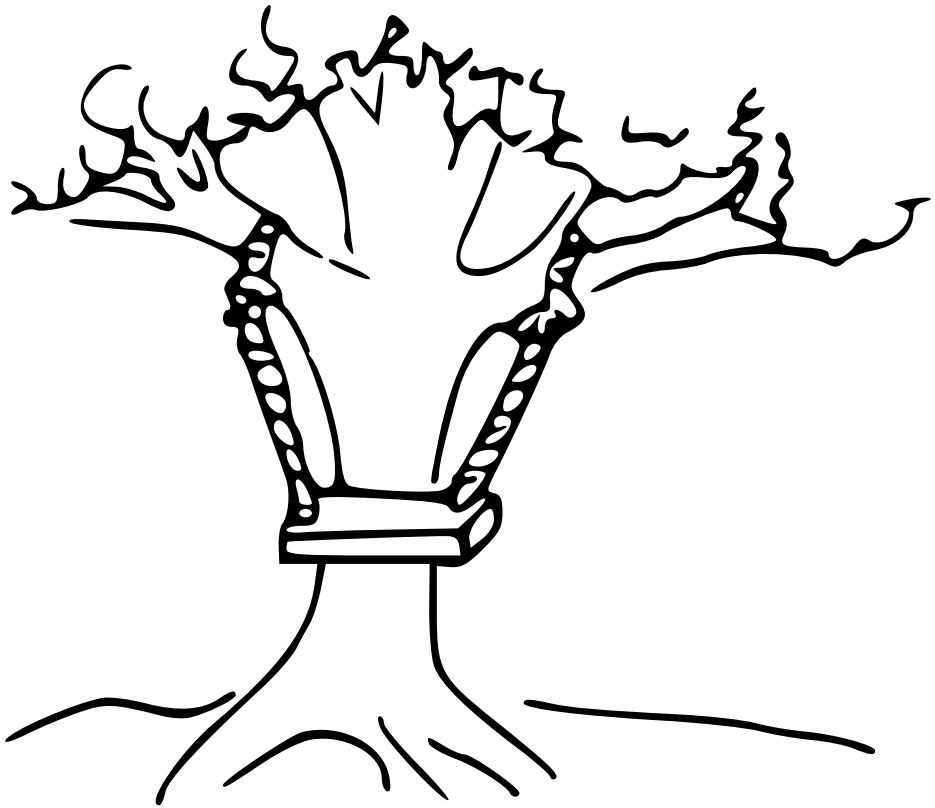
How it was described in the specification:



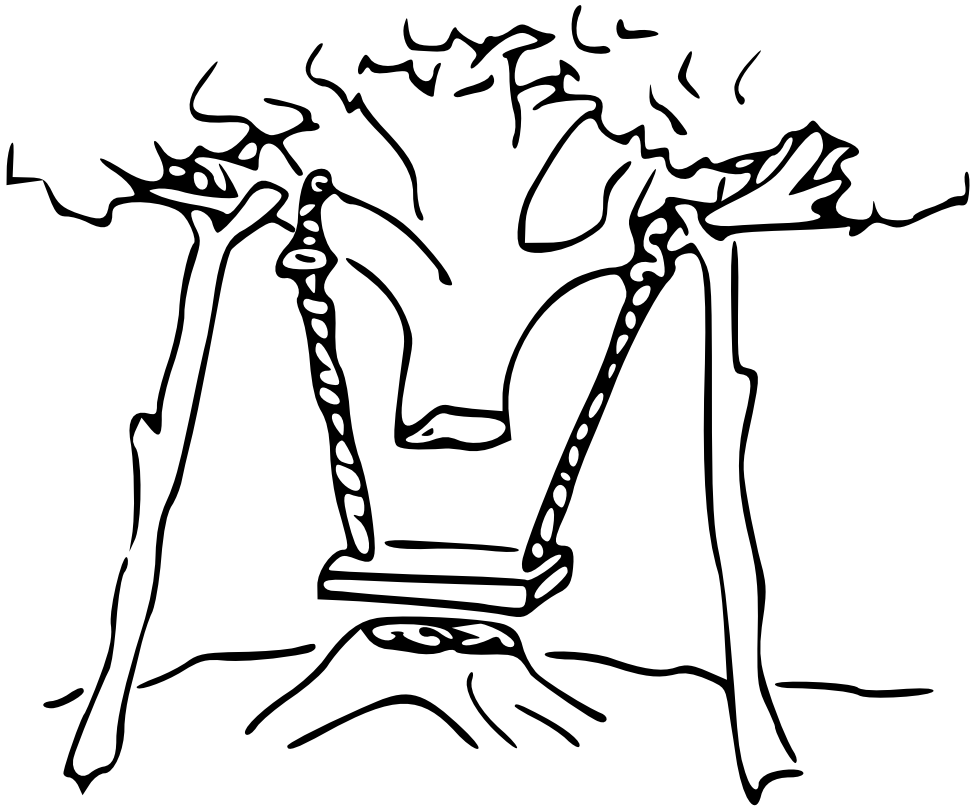
How the lead designer designed it:



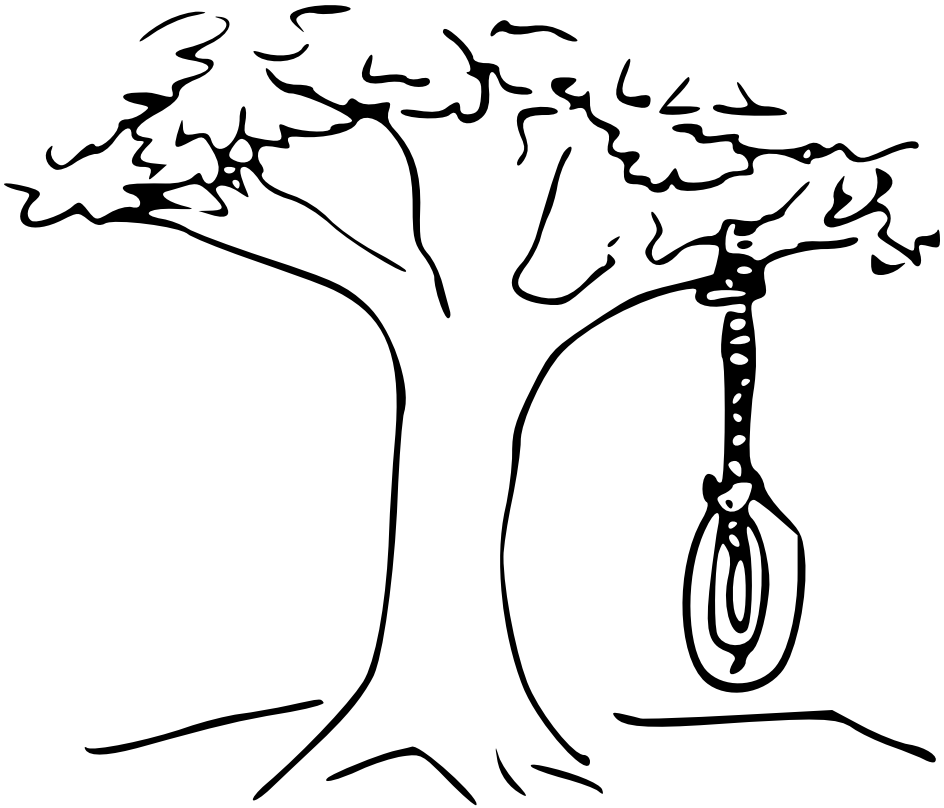
How the programmer implemented it:



How it was finally put into operation:



What the customer really wanted:



Rough procedure:

1. Identification of organizational units
2. Identification of the tasks to be supported
3. Requirements collection plan
4. Requirements collection
5. Filtering
6. Sentence classification
7. Formalization

University employees

Count: 1000

Attributes

- **PersonnelNumber**
  - Type: char
  - Length: 9
  - Domain: 0...999.999.99
  - Number of repetitions: 0
  - Definedness: 100
  - Identifying: yes
- **Salary**
  - Type: decimal
  - Length: (8,2)
  - Number of repetitions: 0
  - Definedness: 90
  - Identifying: no
- **Rank**
  - Type: text
  - Length: 4
  - Number of repetitions: 0
  - Definedness: 100
  - Identifying: no

Relationship Description: *test*

- Participating objects:
  - Professors as examiner
  - Students as examinee
  - Lectures as exam subject
- Attributes of the relationship:
  - Date
  - Time
  - Grade
- Count: 100,000 per year

Process Description: *Issuing Transcripts*

- Frequency: semi-annually
- required data
  - exams
  - study regulations
  - student information
  - ...
- Priority: high
- Amount of data to be processed
  - 500 students
  - 3000 exams
  - 10 study regulations

Specification

- Here the *what* is defined (not the *how*)
- Components of the specification are:
  - process descriptions
  - use cases
  - user interface
  - other interfaces (to other systems)

Specification(2)

- The ideal specification is
  - unambiguous
  - complete
  - understandable (for all readers)
  - free of redundancy
  - …and unattainable in reality

Specification Methods

- There is a large selection of methods:
  - Structured analysis
  - Clerk method
  - Object-oriented methods
  - …
- But not every method is always appropriate

Creating a Specification

- The actual analysis is an iterative (and often very political) process
  - The user tells the developer what they would like
  - The developer writes down everything (that they understood) in their own “language” …
  - …and translates it into the “language” of the user
  - This is shown to the user, with the result that much is still not right
  - Change requests are recorded
  - Back to the second step

Modeling the Data

- Particularly important for database systems is the modeling of the data that is to be stored in the database
- This is handled separately in the conceptual design

- The entity-relationship model (ER model) is frequently used for conceptual design
- An ER schema is a graphical representation of the conceptual modeling of the data
- Identifies *entities* and *relationships* between entities of an application domain

Conceptual Design(2)

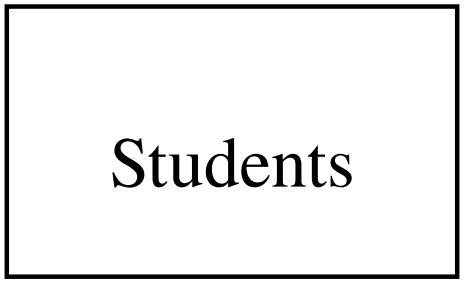
- An ER schema
  - contains all information units that are to be contained in the DBS to be implemented
  - also serves for communication with the future users (to ensure completeness and correctness)
  - can be transformed (relatively) easily into a logical schema

Basic Building Blocks

- Entity type
- Attribute
- Key
- Relationship (relationship type)
- Role

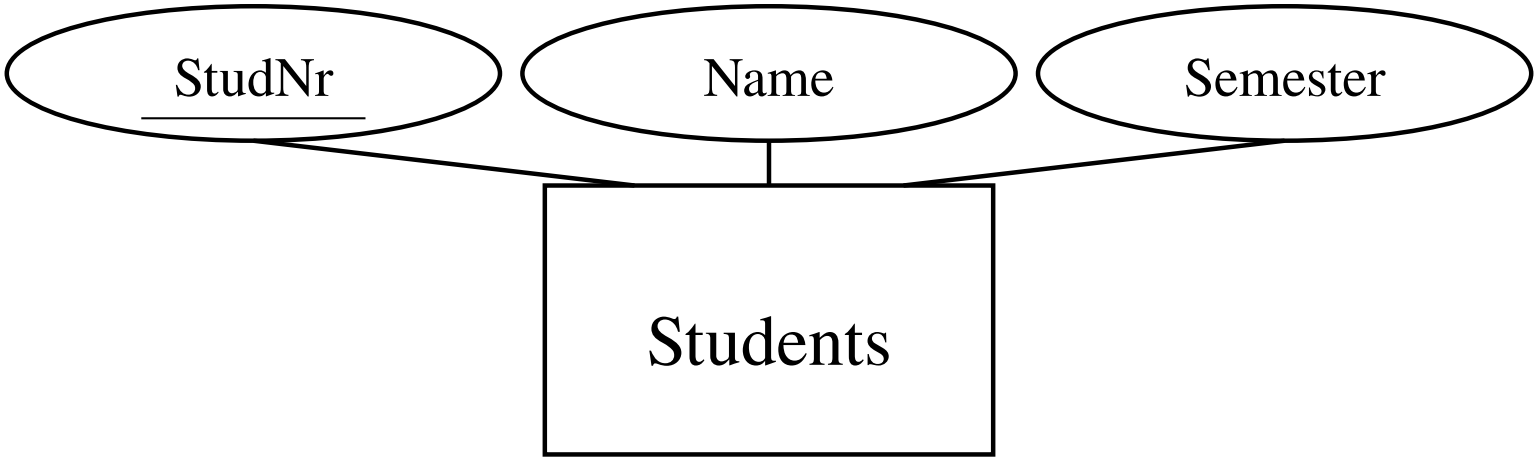
Entities

- Entity: an object of the modeled real-world mini-world that is distinguishable from other objects (e.g. things, persons, concepts)
- Entity types are represented by rectangles:



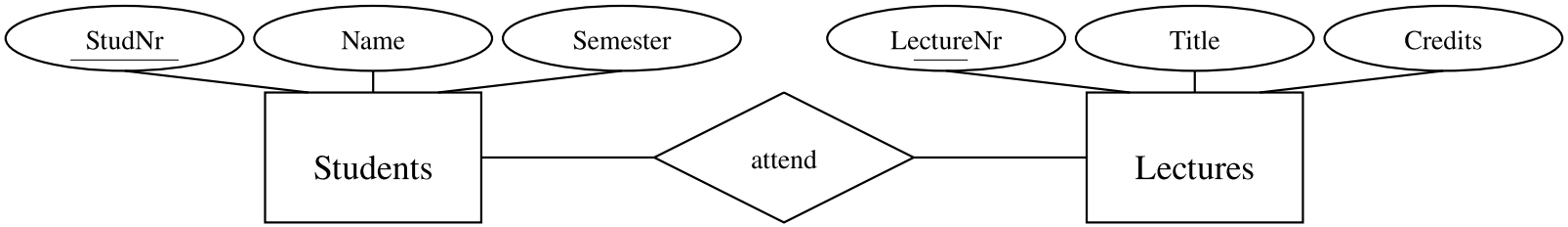
Attributes

- An entity is described by attributes
- Each attribute has a domain (e.g. integer, text) that describes the possible values of this attribute
- An attribute that uniquely identifies an entity is called a *key*
- Attributes are represented by an ellipse:



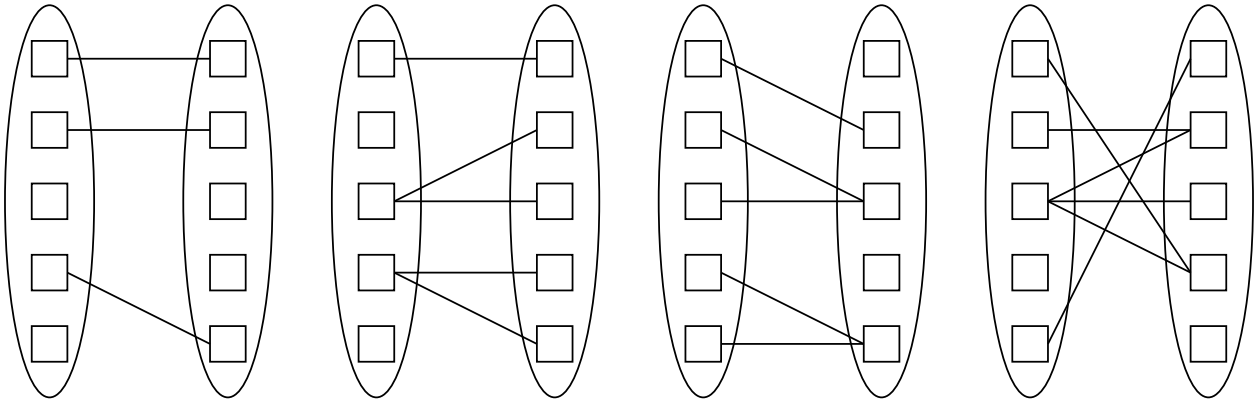
Relationships

- Relationship: connects two or more entity types with each other
- Represented by a diamond:



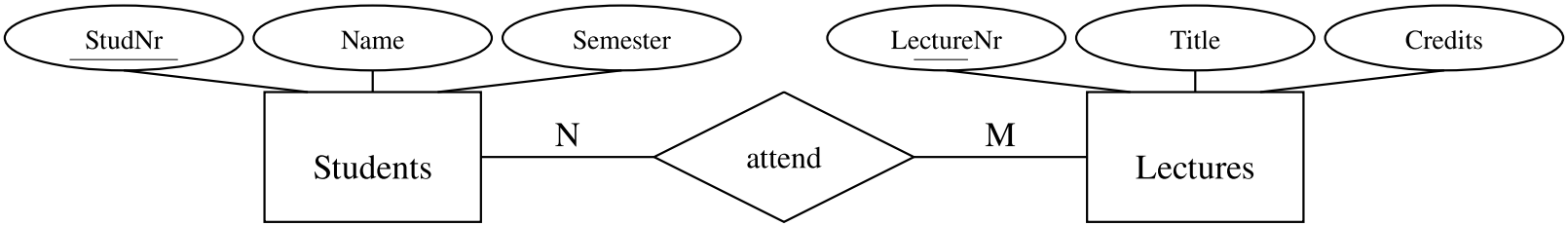
Functionalities

- Further description of relationships by functionalities
- Functionalities express with how many other entities an entity enters into a relationship

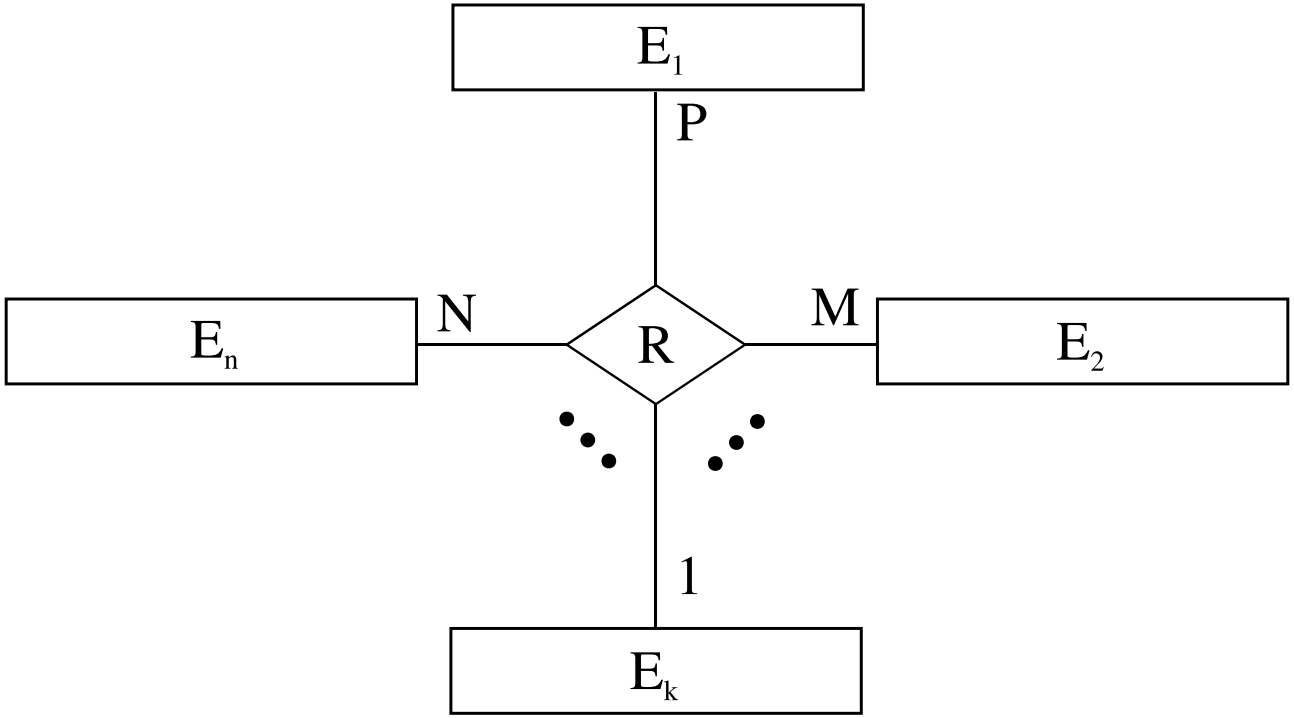


Functionalities(2)

- `attend` is an N:M relationship:

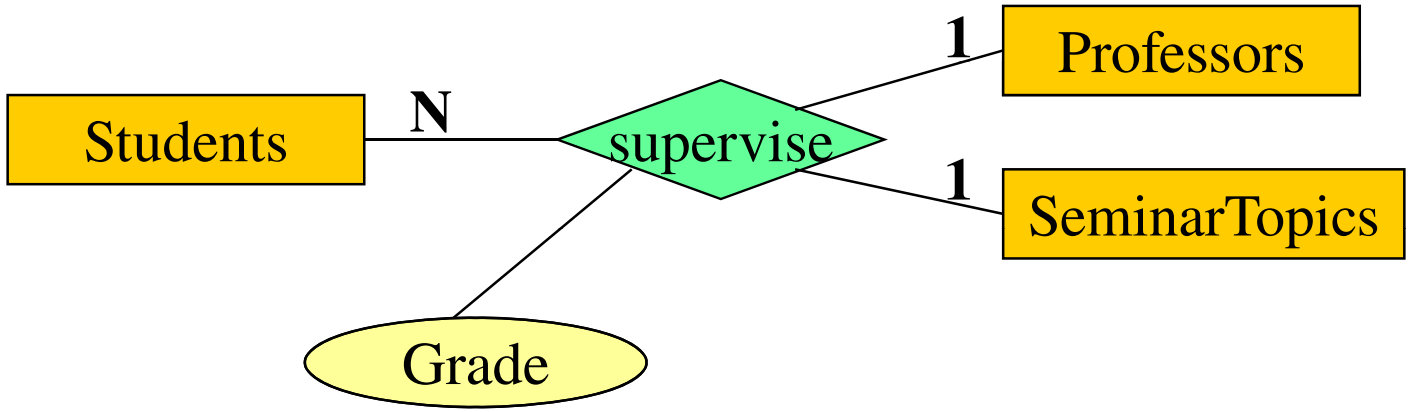


Functionalities(3)



$$R : E_1 \times ... \times E_{k-1} \times E_{k+1} \times ... \times E_n \rightarrow E_k$$

Example relationship: *supervise*



$$supervise : Professors \times Students \rightarrow SeminarTopics$$

$$supervise : SeminarTopics \times Students \rightarrow Professors$$

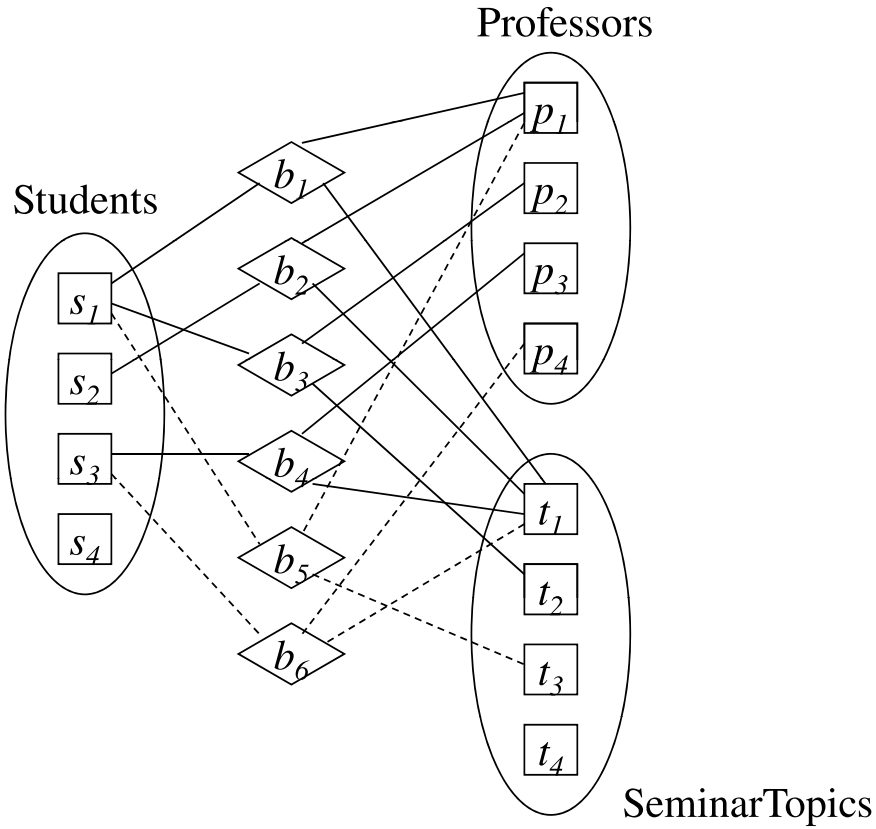
Consistency constraints enforced by this:

1. Students may “complete” only one seminar topic with the same professor (so that a broad spectrum is covered).
2. Students may work on the same seminar topic only once – i.e., they may not work again on a seminar topic already assigned to them with a different professor.

However, the following database states are still possible:

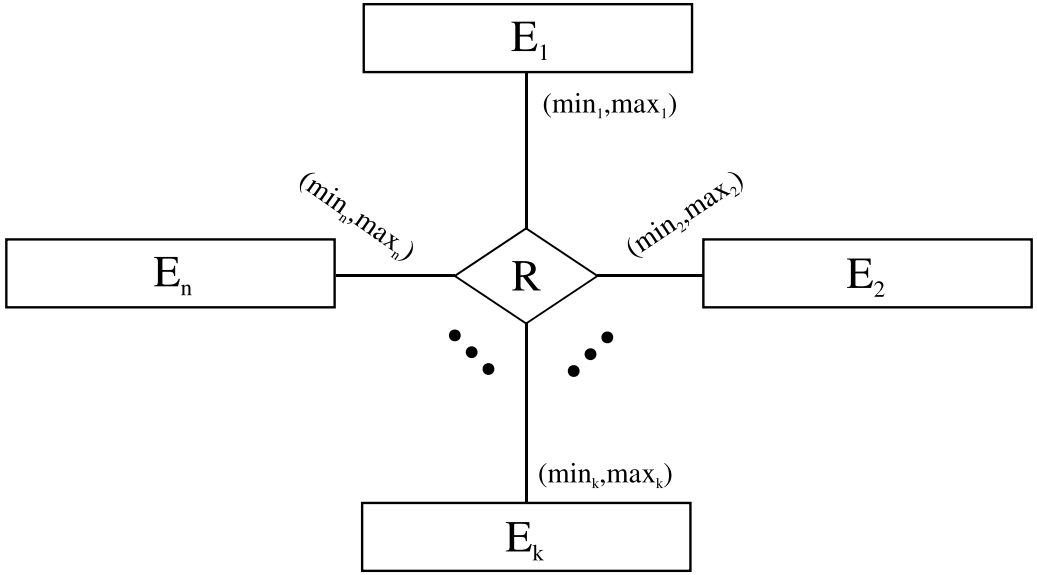
- Professors can “reuse” the same seminar topic – i.e., assign the same topic to several students.
- A topic can be assigned by several professors – but to different students.

Functionalities(6)



Dashed lines mark illegal instances.

Functionalities(7)

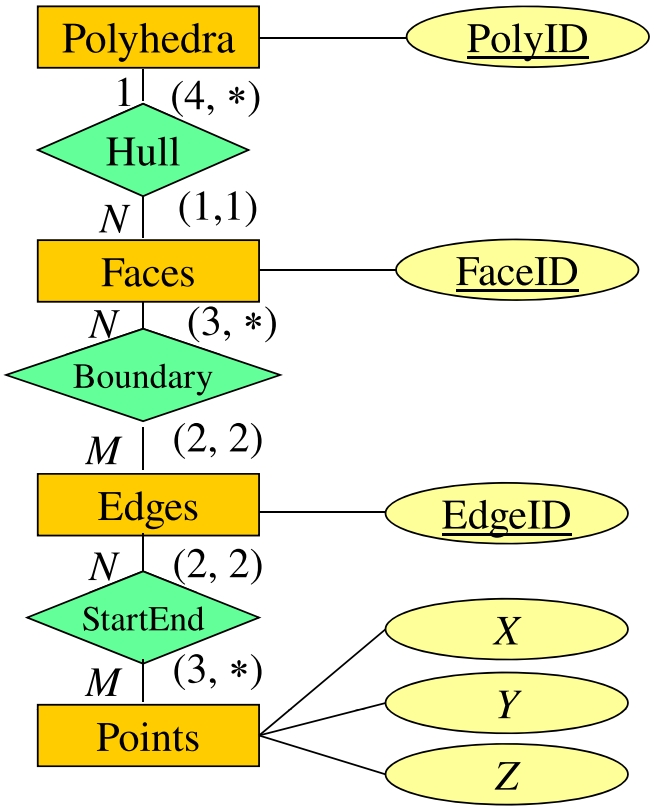


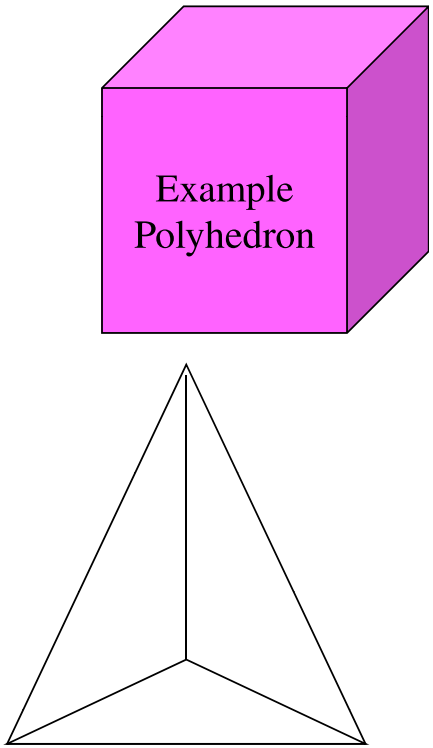
$$R \subseteq E_1 \times ... \times E_i \times ... \times E_n$$

For each $e_i \in E_i$ there are

- at least $\min_i$ tuples of the form $(..., e_i, ...)$ and
- at most $\max_i$ tuples of the form $(..., e_i, ...) \in R$

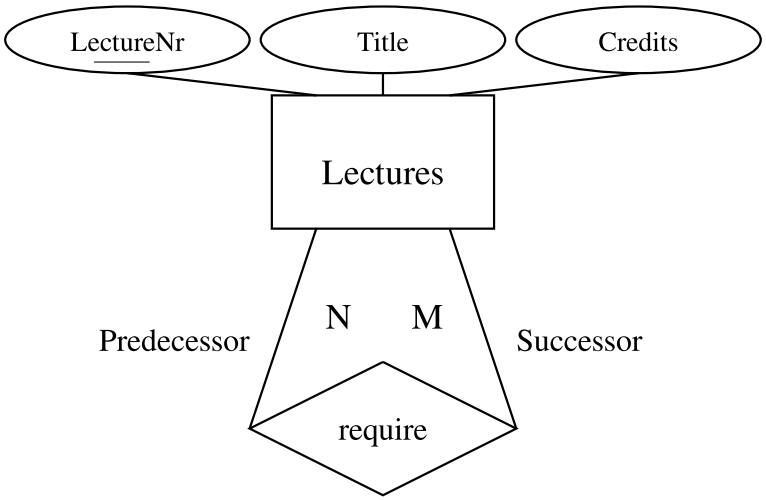
Functionalities(8)



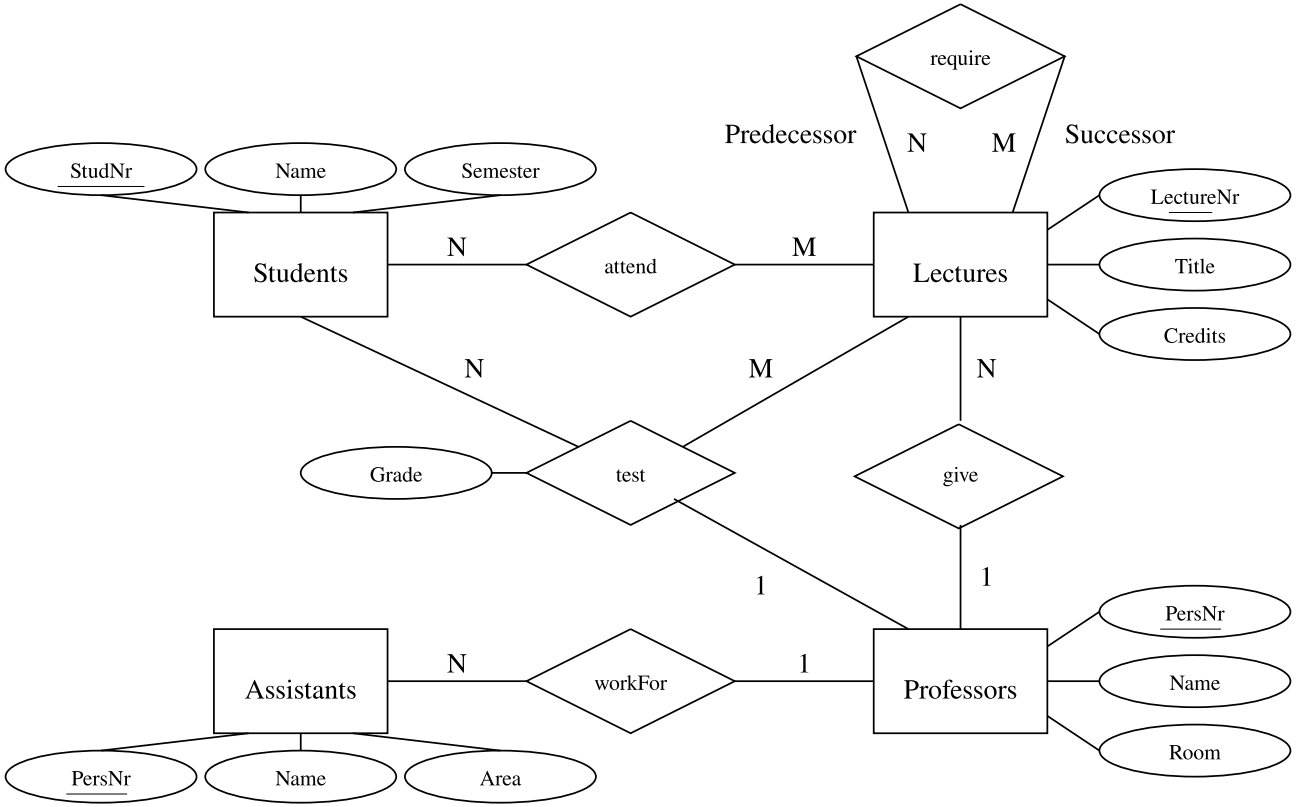


Roles

- Roles describe in which way entities participate in a relationship
- Especially useful for describing recursive relationships



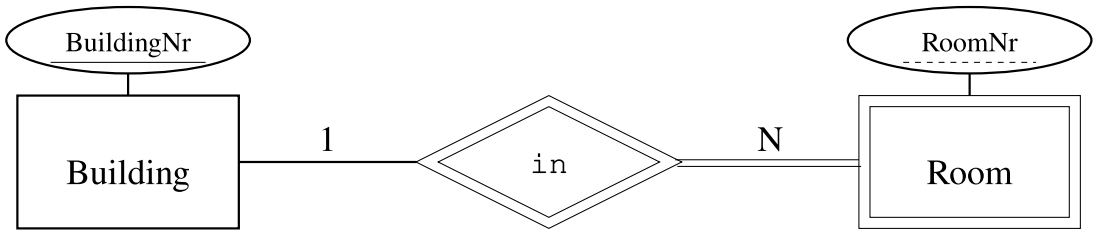
Complete Schema



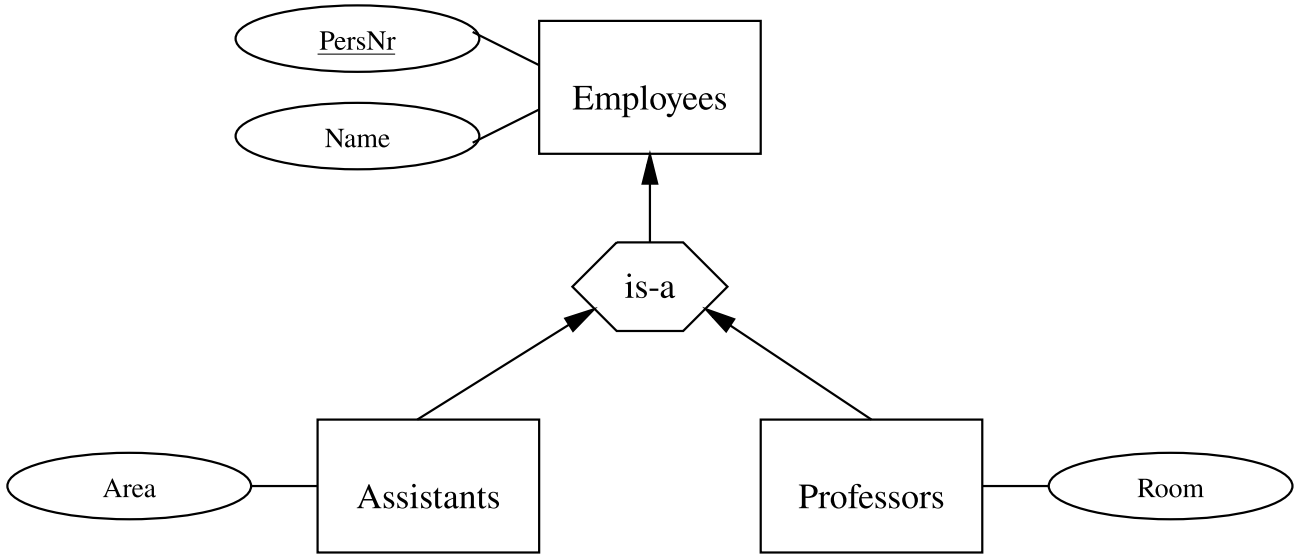
- Weak entities
- Generalizations
- Aggregations

Weak Entities

- Weak entity: cannot exist as an independent object
- Needs a “strong” entity in order to be uniquely identified
- A weak entity and its associated strong entity enter into a 1:N (or, more rarely, 1:1) relationship

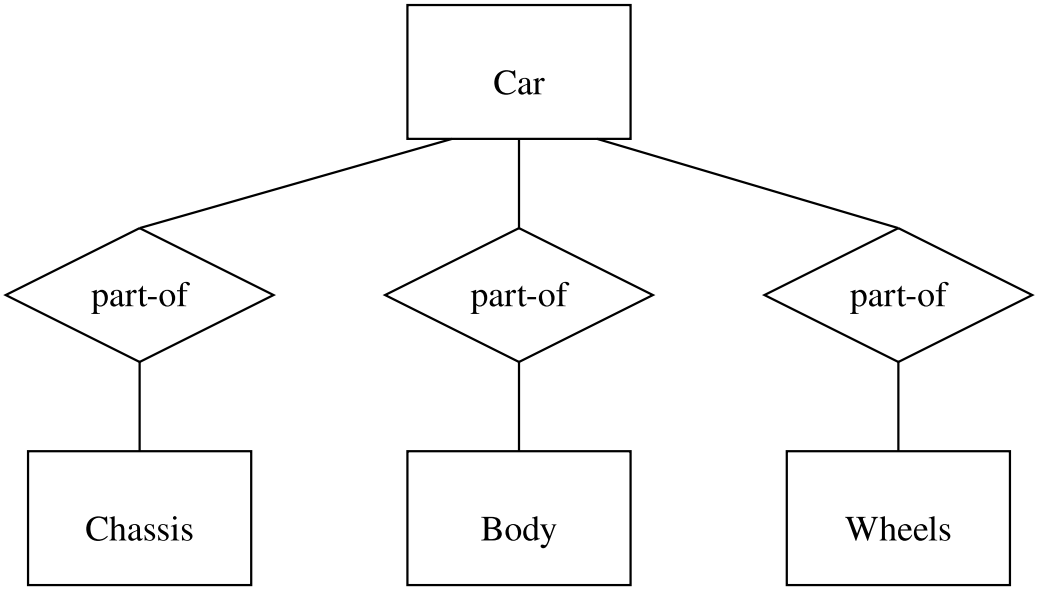


- As in object-oriented programming languages, attributes can be inherited from a supertype by a subtype



Aggregation

- Used for modeling is-part-of relationships



Design Decisions

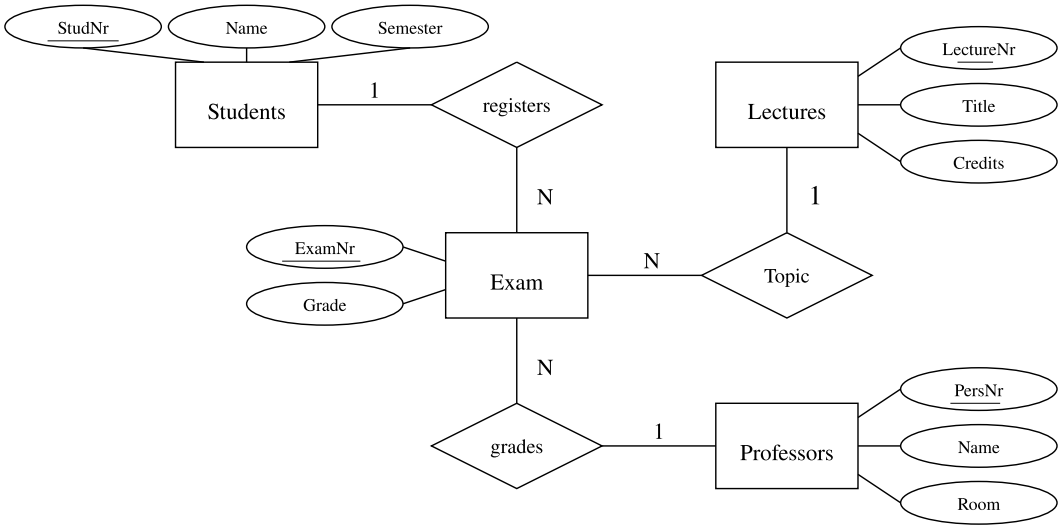
- Entity vs. attribute
- Entity vs. relationship
- Binary vs. ternary relationships
- Connection trap

Entity vs. Attribute

- A separate entity type is more flexible, an attribute simpler
- Example: we want to store address information of customers
  - If there can be multiple addresses per customer, model address as a separate entity (since attributes may not be set-valued)
  - If an address is to be split into individual components (street, house number, ZIP code, city, etc.), then decompose address into individual attributes (these can then possibly be moved out into a separate entity)

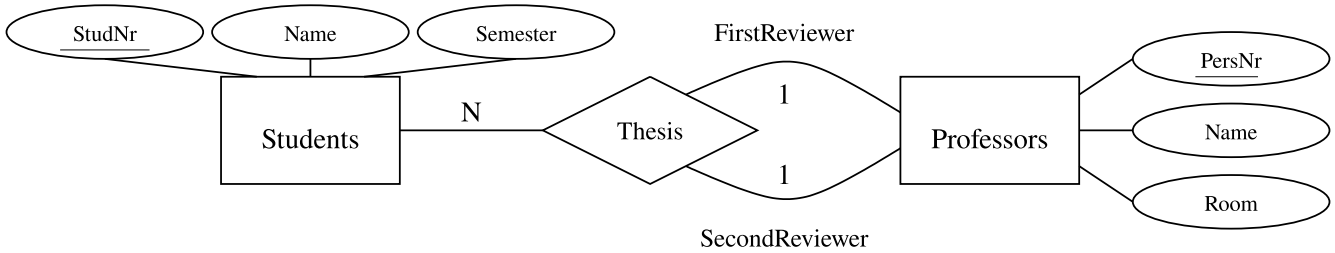
Entity vs. Relationship

- Entity is more flexible, relationship simpler
- Relationships have no identity of their own: the same entities cannot enter into a relationship more than once
- Example: more than one grade for the same lecture from the same professor:



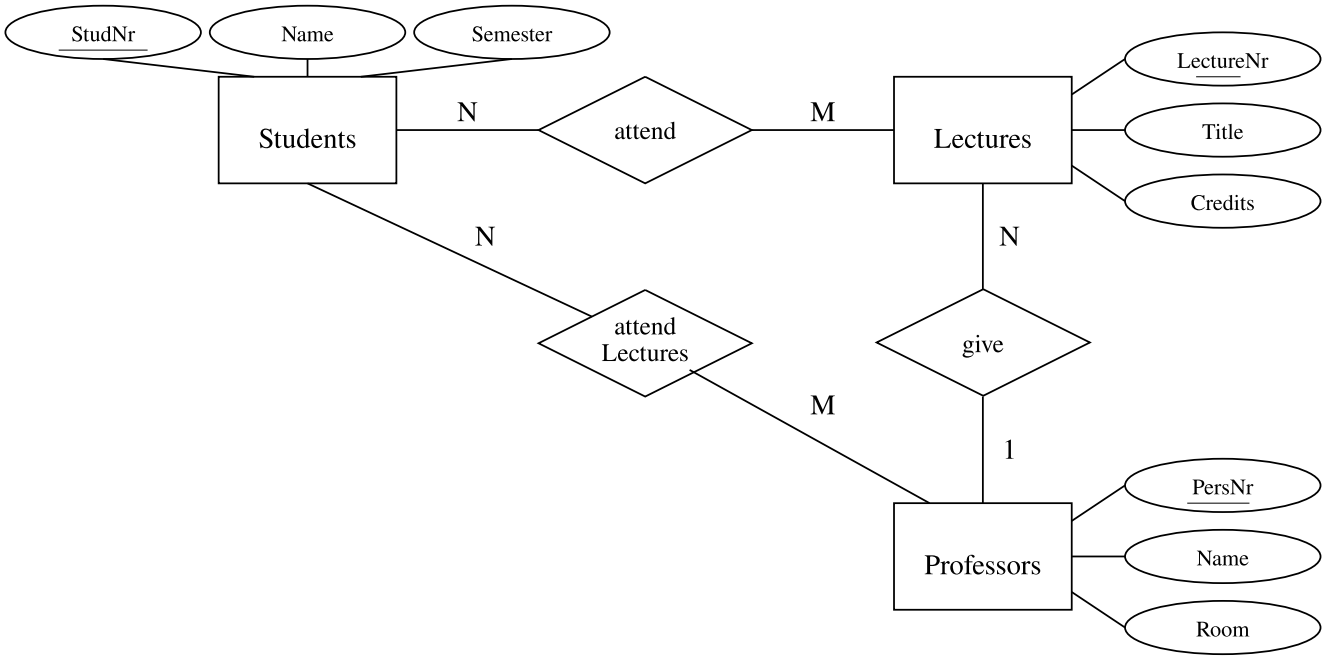
Binary vs. Ternary Relationship

- Genuine ternary relationships need all three participating entities to describe the relationship
- Relationship `test` in the complete schema is genuinely ternary
- The following relationship is not:



Connection Trap

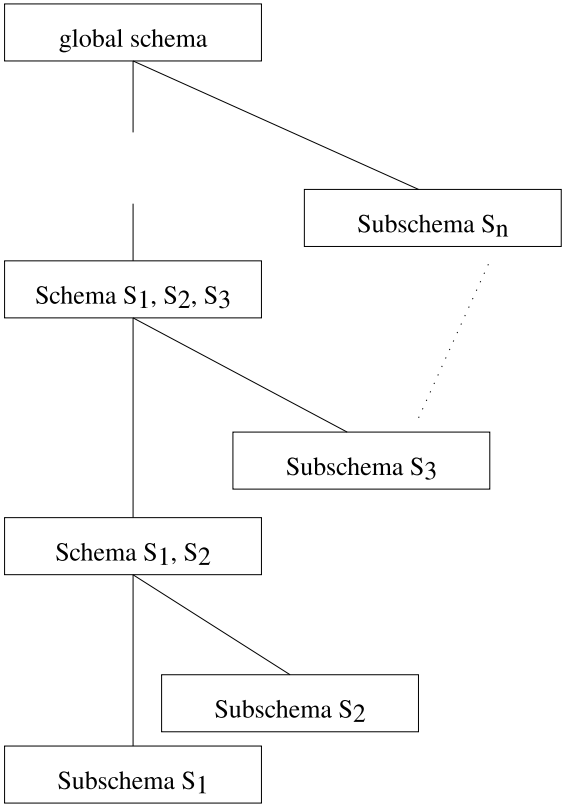
- A trap that is easy to fall into is the connection trap: redundant, cyclic relationships
- `attend Lectures` is redundant, it is already expressed by `attend` and `give`



Schema Consolidation

- Applications are often modeled by different people who also have different viewpoints on the application
- This results in subschemas that have to be merged into an overall schema

Schema Consolidation(2)

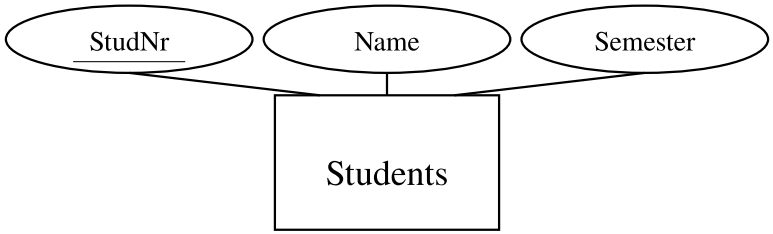


Translation ER → Relational

- For a relational DBMS, an ER schema has to be translated into a relational schema
- This happens in three steps:
  - translation of the entities
  - translation of the relationships
  - simplification of the schema

Translation of Entities

- Convert each entity into a relation
- Each attribute of the entity becomes an attribute of the relation (key of the entity becomes primary key of the relation)



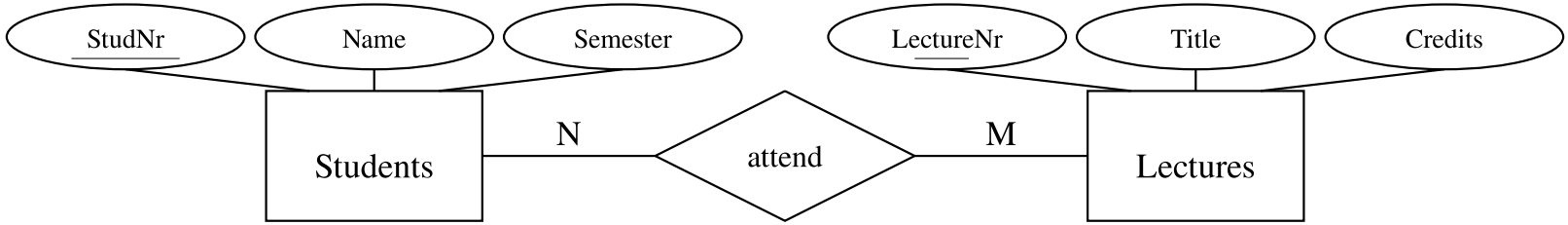
$\Downarrow$

Students(<u>StudNr</u>, Name, Semester)

Translation of Relationships

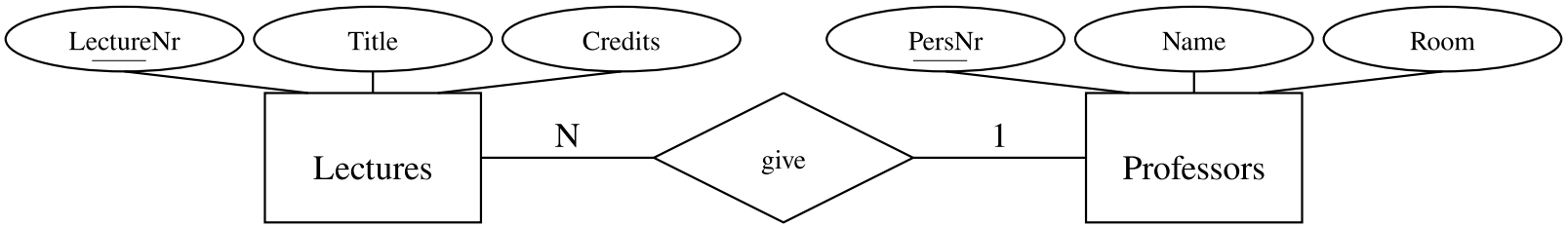
- Convert each relationship into a relation
- Attributes of this relation are the keys of the participating relations (+ attributes of the relationship itself)
- The key of a (binary) relationship depends on the kind of relationship
  - N:M - key is composed of the combination of both entity keys
  - 1:N, N:1 - key of the N side becomes the key
  - 1:1 - either of the two entity keys can be chosen

Examples



$$\Downarrow$$

attend(<u>StudNr, LectureNr</u>)



$$\Downarrow$$

give(<u>LectureNr</u>, PersNr)

Schema Simplification

- After the second step, relations with the same key can exist
- These relations are merged into a single relation
- This happens for 1:N, N:1, and 1:1 relationships, since the relationship relation has the same key as one of the entity relations

Example

- `Lectures` and `give` have the same key:

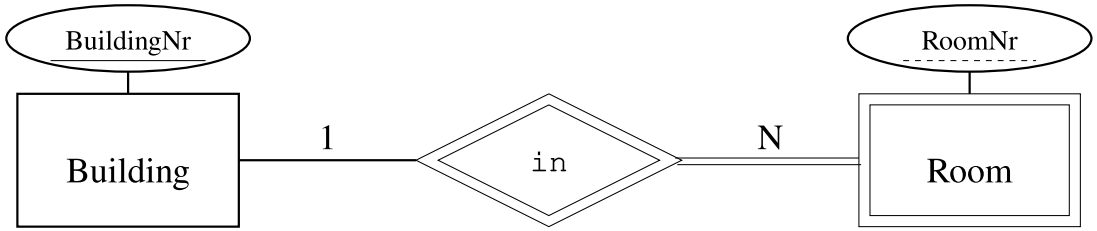
$$\text{Lectures}(\underline{\text{LectureNr}}, \text{Title}, \text{Credits})$$
$$\text{give}(\underline{\text{LectureNr}}, \text{PersNr})$$
$$\Downarrow$$
$$\text{Lectures}(\underline{\text{LectureNr}}, \text{Title}, \text{Credits}, \text{PersNr})$$

- `PersNr` is called a *foreign key* of `Lectures`, since it references the key in `Professors` (and also contains the same values)
- Often renamed to make references clear

$$\text{Lectures}(\underline{\text{LectureNr}}, \text{Title}, \text{Credits}, \text{GivenBy})$$

Weak Entities

- Weak entities are in principle translated like 1:N relationships
- Only difference: the weak entity has a composite key

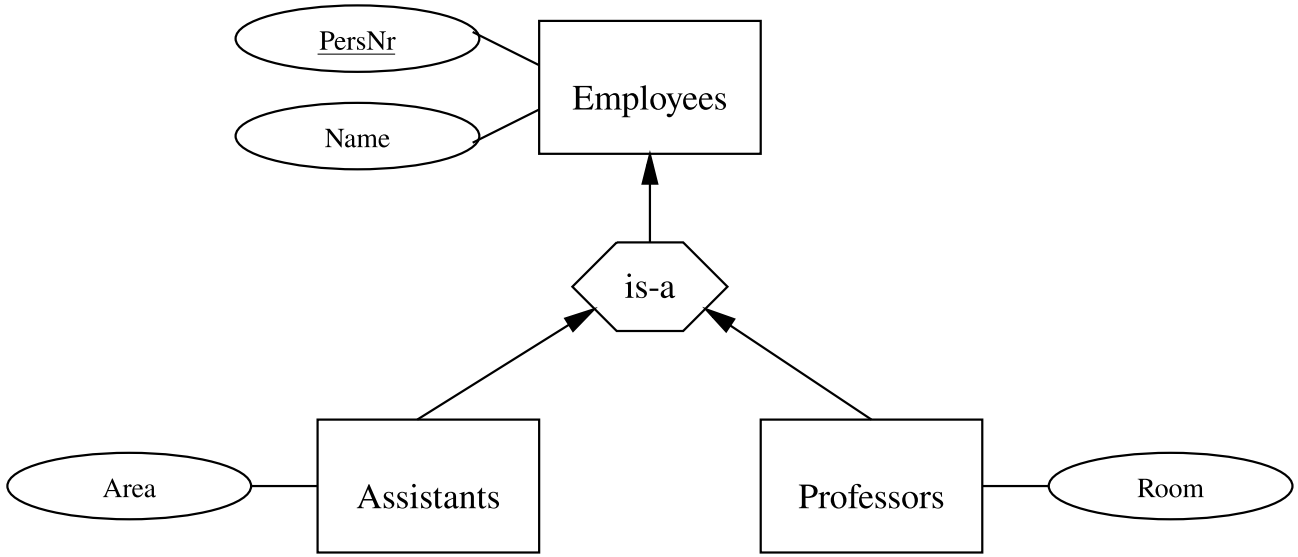


$$\Downarrow$$

Building(<u>BuildingNr</u>, ...) Room(<u>BuildingNr, RoomNr</u>, ...)

Generalizations

- Generalizations can be translated in three different ways (there is no directly equivalent translation into the relational model)



$\Downarrow$

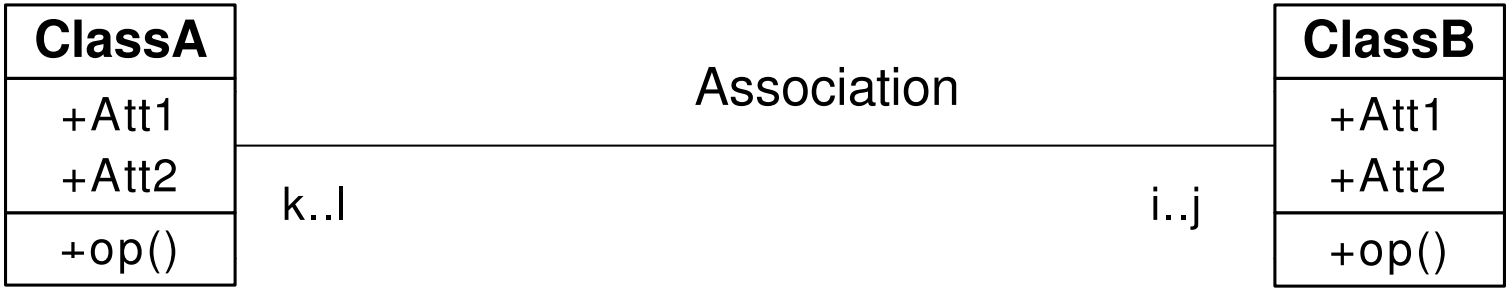
(1) Only supertype, (2) only subtypes, (3) all types

Generalizations(2)

- Only supertype:
  Employees(PersNr, Name, Area, Room)
  Problem: NULL values
- Only subtypes:
  Assistants(PersNr, Name, Area)
  Professors(PersNr, Name, Room)
  Problem: employees who are neither assistants nor professors
- All types:
  Employees(PersNr, Name)
  Assistants(PersNr, Area)
  Professors(PersNr, Room)
  Problem: information is distributed over multiple relations

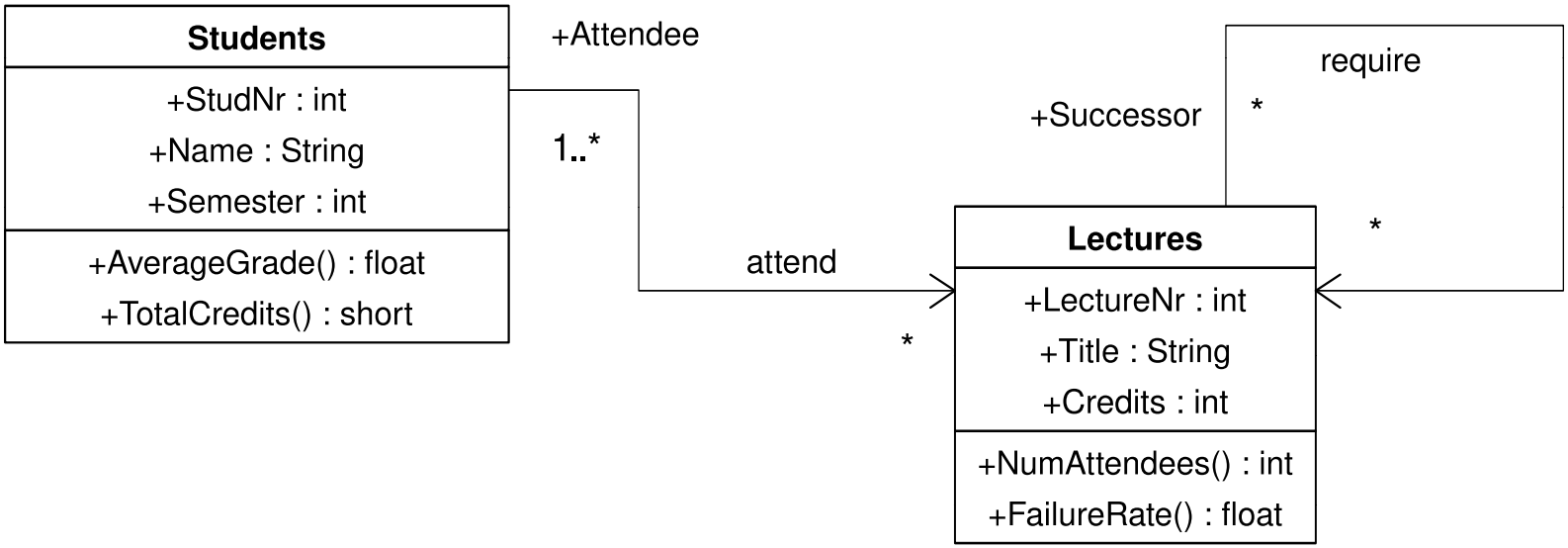
- De facto standard for object-oriented software design
- Central construct is the class, with which similar objects are modeled with respect to their
  - structure (attributes)
  - behavior (operations/methods)
- Associations between classes correspond to relationship types
  - generalization hierarchies
  - aggregation

Multiplicities

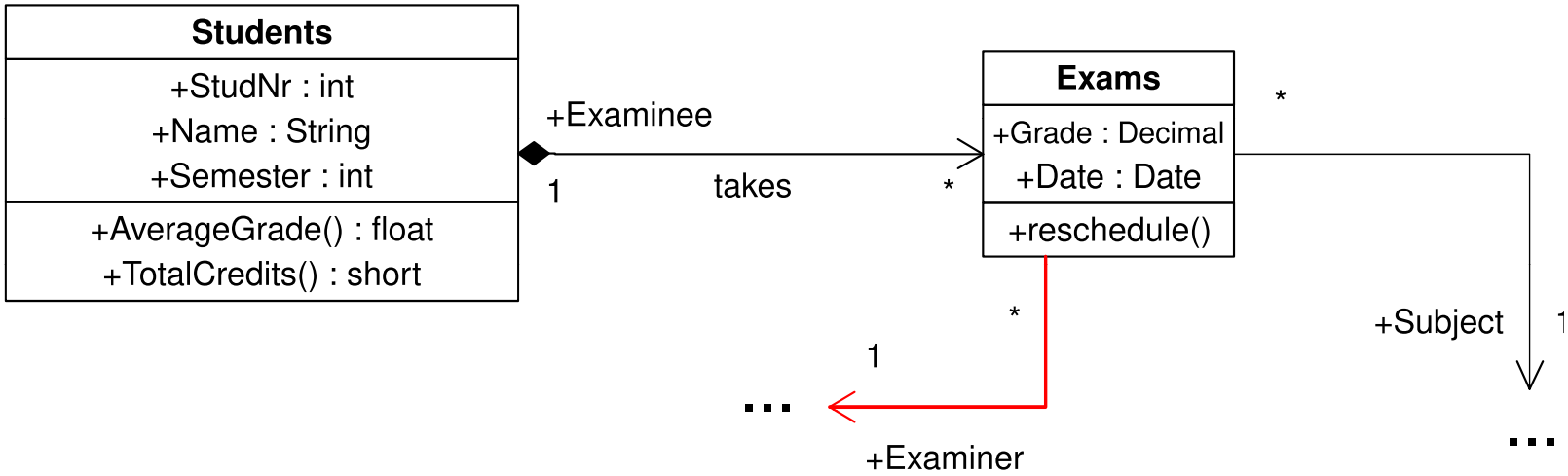


- Each element of ClassA is related to at least i elements of ClassB
- … and to at most j ClassB elements
- The same holds analogously for the interval k..l
- The multiplicity specification is analogous to the functionality specification in the ER model
  - Not to the (min,max) specification: **Caution!**

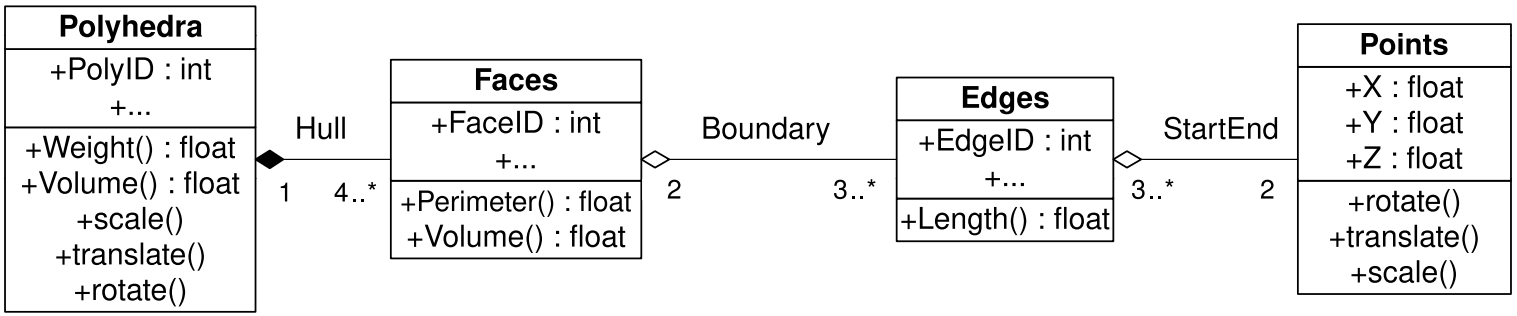
Classes and Associations

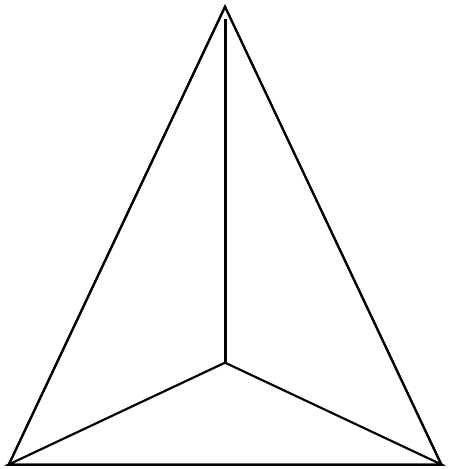


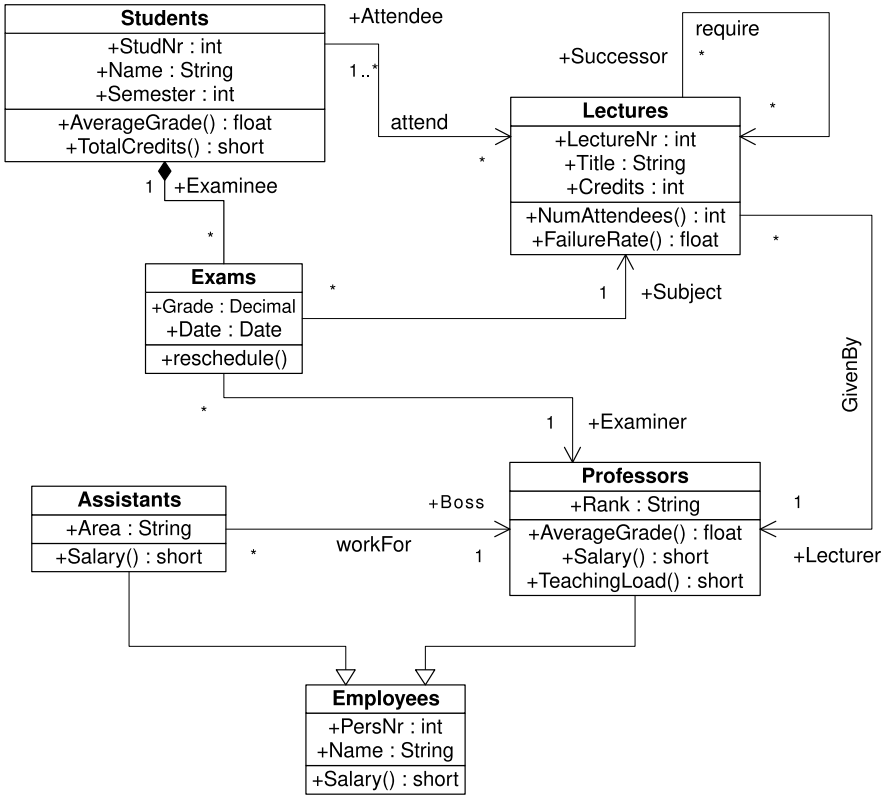
Aggregation



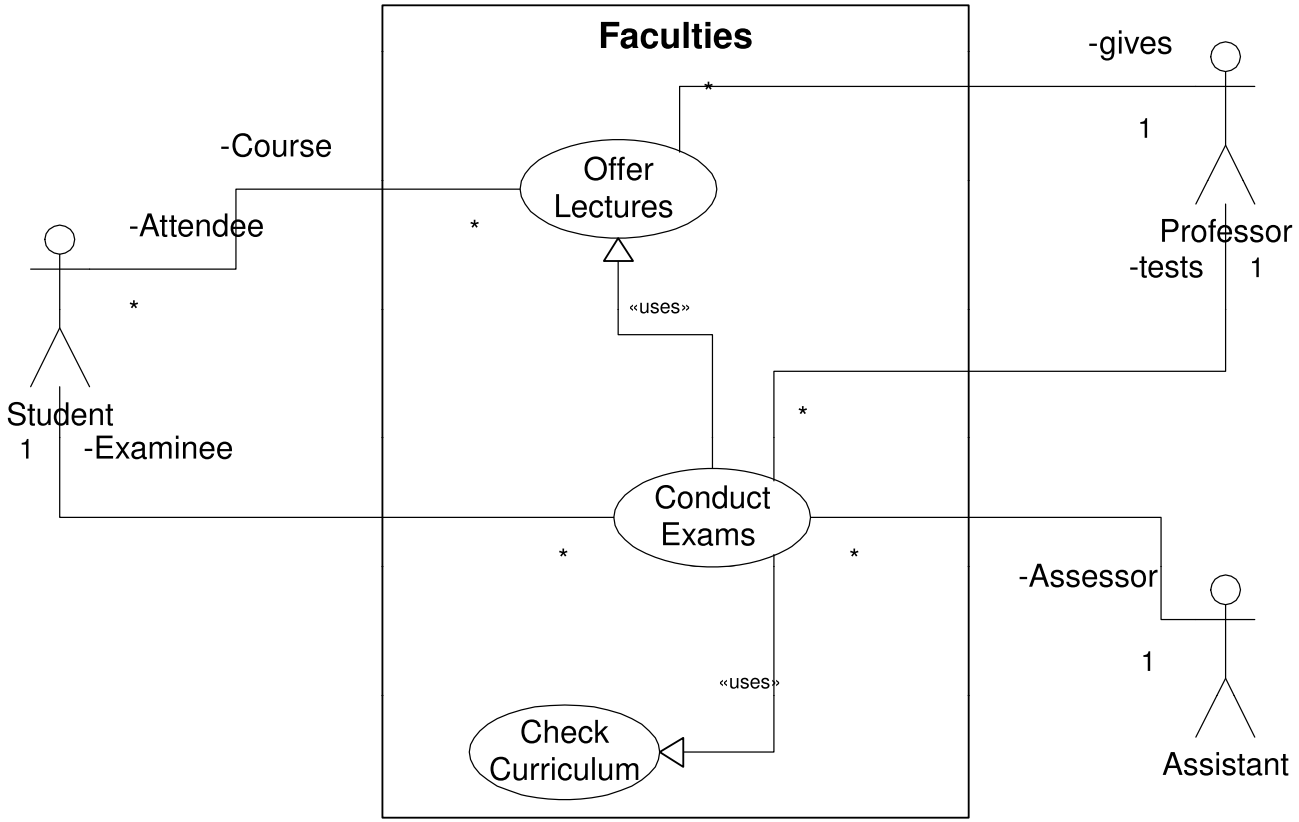
Boundary Representation Modeling in UML



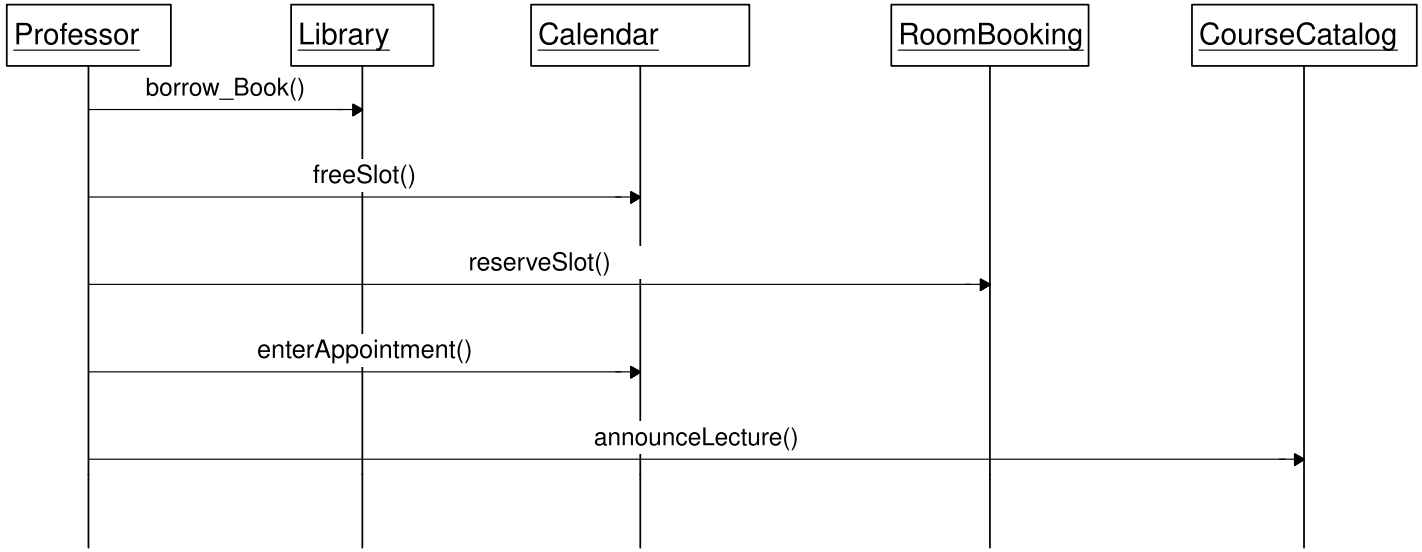



Students
+StudNr : int
+Name : String
+Semester : int
+AverageGrade() : float
+TotalCredits() : short
+Attendee
1..*
attend
*
Lectures
+LectureNr : int
+Title : String
+Credits : int
+NumAttendees() : int
+FailureRate() : float
+Successor
require
*
*
1
+Examinee
*
Exams
+Grade : Decimal
+Date : Date
+reschedule()
*
1
+Subject
*
*
1
+Examiner
GivenBy
Professors
+Rank : String
+AverageGrade() : float
+Salary() : short
+TeachingLoad() : short
1
+Lecturer
Assistants
+Area : String
+Salary() : short
+Boss
*
workFor
1
Employees
+PersNr : int
+Name : String
+Salary() : short

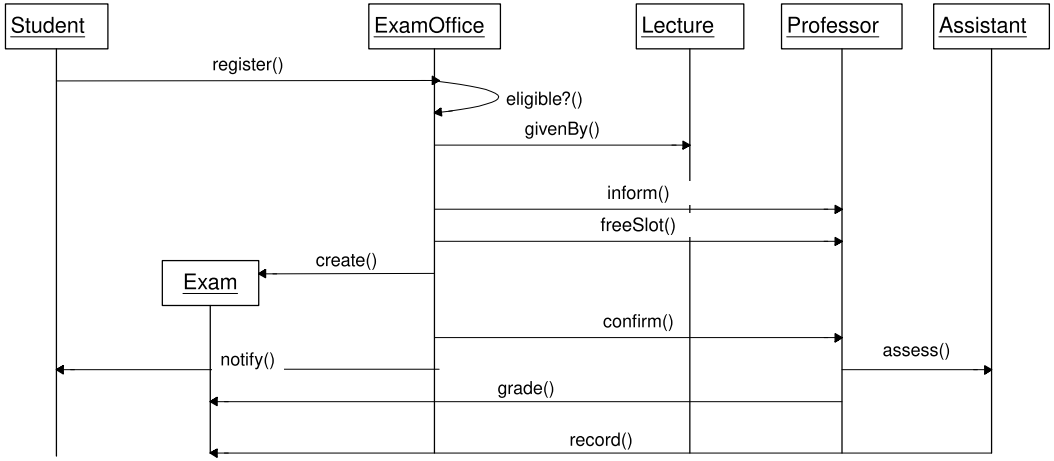
Use Cases


Faculties
-gives
*
1
-Course
Offer
Lectures
-Attendee
*
*
Professor
-tests
1
«uses»
Student
1
-Examinee
*
Conduct
Exams
*
*
-Assessor
1
«uses»
Check
Curriculum
Assistant

Interaction Diagram



Interaction Diagram *Conducting an Exam*



Summary

- ER and UML are widely used kinds of conceptual modeling
- The concepts are not too hard to grasp, i.e., the subject matter can also be discussed with non-computer scientists (who know the application domain)
- Design is always somewhat subjective, there are many different ways to model a situation (choosing the most suitable one is a matter of experience)
- The quality of a conceptual schema can be checked once more at the logical level
- The conceptual model is finally transformed into the relational model