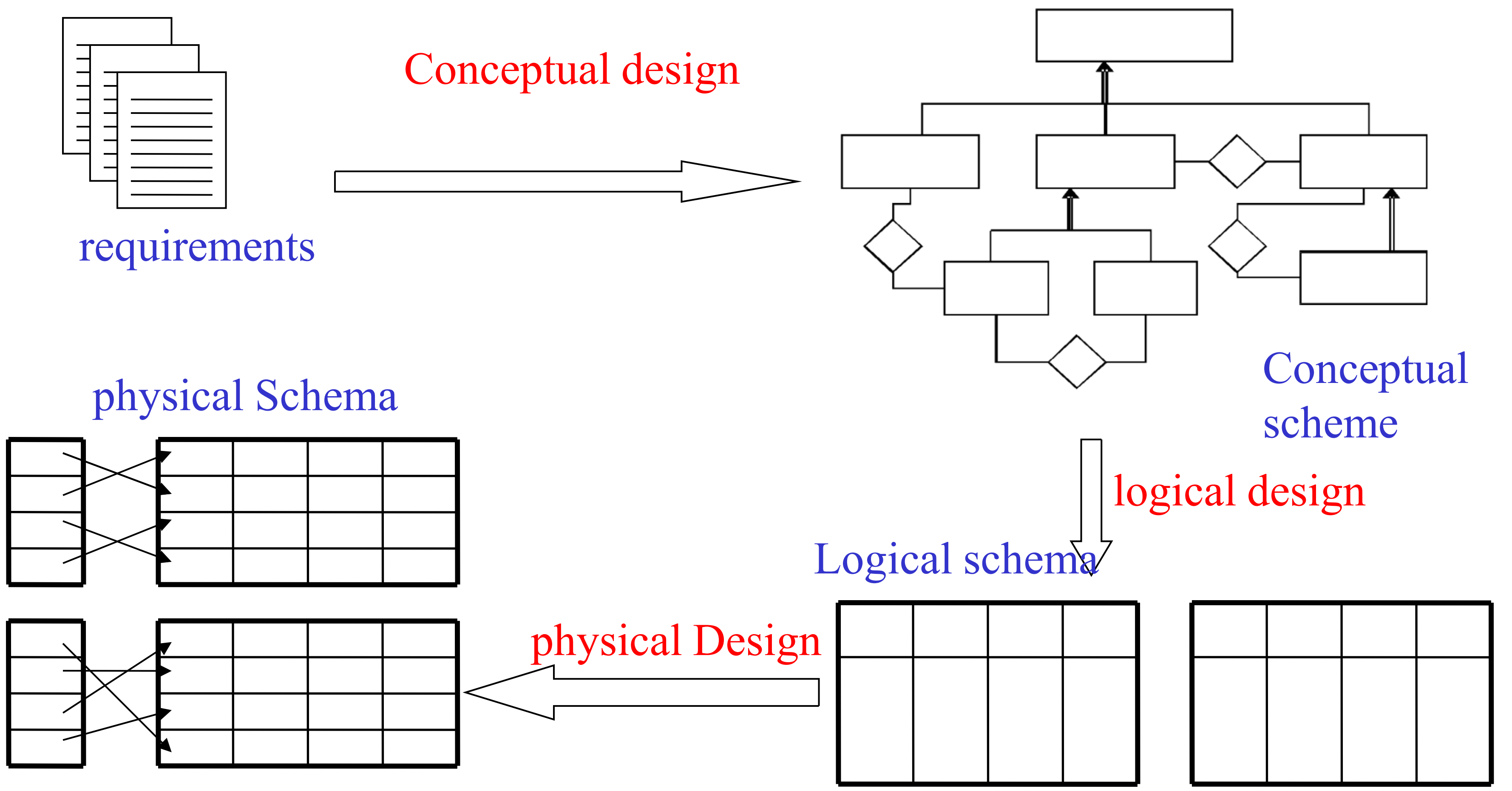
A design methodology for databases

It is divided into three main phases, to be carried out one after the other, and is based on the principle of ***abstraction*** applied to programming: separate decisions on "what" to represent in a database (*first phase*), from those relating to "how" to do it (*second* is *third phase*).

A methodology for databases design



The products of the various stages of the design of a relational database are made with the Entity-Relationship model

The conceptual design

The conceptual design of a database involves the construction of an ER diagram which can best describe the reality of interest.

The schema construction is a gradual process, as it complex also for relatively small databases.

The construction of the conceptual model is preceded by two activities:

- *Requirements Gathering*
- *Requirements Analysis*

The requirements gathering

For ***requirements gathering***, which means a complete identification

1. of the *issues* that the system must resolve
2. the *features* that such a system must have
   - 2.1. *static* aspects (Data)
   - 2.2. *dynamic* aspects (The operations on the data)

The requirements gathering

Possible sources for the requirements are:

- users, through:
  - interviews
  - appropriate documentation
- existing documentation:
  - regulations (laws, industry regulations)
  - internal regulations, internal procedures
- existing applications:
  - organizing data
  - features implemented
  - attention ... it is often best to start from scratch!

The requirements gathering

The process of gathering requirements is a difficult activity and not standardized, mainly because requirements are almost always expressed in natural language (must be interpreted according to the context and can be ambiguous)

In the case of gathering through interviews, it should be taken into account that:

- different users can provide different information (although apparently ...)
- higher-level users often have a wider but less detailed view

Interviews often lead to an acquisition of the requirements by "continuously refining them"

The requirements gathering

A few rules to keep in mind in the acquisition of requirements through an interview:

- Continuously check understanding and consistency (especially compared to the used terminology)
- Verify by means of examples (general, but also concerning borderline cases)
- Highlight the essential aspects with respect to marginal ones (but this can change depending on the type of users and the tasks they have to perform)

The requirements gathering

An example of requirements collected for the development of a database for a training company is as follows:

Education company
We want to create a database for a company that supplies courses, We want to represent data of trainees and teachers. For the participants (5000), identified by a code, we want to store the tax code, last name, age, sex, place of birth, the name of the company for which they work at present, places where they previously worked and the time, together with the address and the telephone number, the courses they have attended (the courses are in all about 200), the final mark and the period. We also represent the seminars they are currently attending and, for each day, the places and the hours where classes are held.

The requirements gathering

Education company
The courses have a code, a title and can have various editions with a number of participants. If the students are professionals, we want to know their area of interest and, if they have, the title. For those who work for state bodies, we want to know their level and position held. For the teachers (300) we represent the name, age, place where they were born, the name of the course they teach, those they have taught in the past and those they can teach. We also represent all their telephone numbers. The teachers may be company employees or external collaborators.

Requirements analysis

Requirements are initially collected in specifications usually expressed in natural language and, for this reason, often ambiguous and disorganized.

Example The text on the requirements on the education company presents ambiguities, such as:

"..., *the places where they worked before ...* "

"...*the place where they were born, ...* "

The ***requirements analysis*** consists in the clarification and organization of requirement specifications.

Requirements analysis

General rules for requirements analysis:

- ***Choose the right level of abstraction***: Avoid choosing terms too abstract or too specific.
  Example: Preferable "*professional title*" to "*title*" or "*mark in tenths*" to "*mark*".
- ***Standardize sentence structure*** Always use the same syntax style.

Example "*For* <data>, *we represent* <property>"

Requirements analysis

- ***Linearize phrases and divide those articulated***.
  Example: Preferable "*state employees*" to "*those working for state bodies*"
- ***Identify homonyms and synonyms***; unify terms.
  Example: distinguishing *place* in *company* and *city*
  But use only *participant* to denote also one *student* and one *trainee*.
- ***Making explicit reference between terms***.
  Example. In the second sentence is not clear whether the terms *address* and *telephone number* are related to the course participants or to the companies for which they work.

Requirements analysis

- ***Building a glossary for terms*.**

Term	Description	Synonyms	Connections
Participant	Participant in courses. It can be an employee or professional	Student, trainee	Course, Company
Instructor	Teacher of the courses. It can be external.	Teacher, lecturer	Course
Course	Courses offered. They may have different editions.	Seminar	Instructor Participant
Company	Company at which participants work and / or have worked	Places	Participant

Requirements analysis

- ***Reorganizing phrases per Keywords*** (we may have to replicate).

general phrases
We want to create a database for a company that supplies courses. We want to represent data of participants and instructors.

Requirements analysis

Phrases for participants
For participants (5000), identified by a code, we want to store the tax code, last name, age, sex, city of birth, the name of the company they now work for and the companies for which they worked in the past (with the period of work), editions of the courses they have attended, with its final mark in tenths, and the period of attendance. We want also to represent the editions of courses they are currently attending.

Phrases related to the company
As regards the companies for which participants are currently working and / or have worked, we represent the name, address and telephone number.

Requirements analysis

Phrases related to courses
For courses (around 200), we represent the title and code, the various editions and, for running editions, we represent the day of the week, the classrooms and the hours where classes were held and the number of participants.

Phrases for specific types of participants
For participants who are professionals, we represent the area of interest and, if they have, the professional title. For participants who are state employees, we represent their level and position held.

Requirements analysis

Phrases related to instructors
For instructors (about 300), represent the name, age, town of birth, all the phone numbers, the title of the course they teach, those they have taught in the past and of those that they can teach. Instructors can be company employees or external collaborators.

Requirements analysis

Separate the sentences on the data from those on the functions. In addition to the specifications on the data, also specifications on the specific operations to be performed on these data and their frequency should be collected (information, this, very useful in the design phase).

- Enter a new participant indicating all his/her data *(40 times a day, on average)*.
- Assign a participant to an edition of the course *(50 times a day)*.
- Enter a new instructor indicating all his/her data and courses that he/she can teach *(2 times a day)*

Requirements analysis

- Assign an enabled instructor to an edition of the course *(15 times a day)*
- Print all information about the current editions of a course: title, class schedules and number of participants *(10 times a day);*
- Print all courses offered, with information on instructors who can teach them *(20 times a day);*
- For every instructor, find the participants in all the courses he / she taught *(5 times a week);*
- Make a statistic of all course participants with all the information about them, the edition which was attended and the respective mark *(10 times a month)*.

Representation issues

There is no single way to represent a set of specifications. However, it is possible to establish some **general criteria**:

- *If a concept has significant property and / or describes an object class with autonomous existence, we should represent it with an entity.*
  Example: *instructor* and *course* will be represented as an entity.

- *If a concept has a simple structure and does not have associated relevant properties, it is appropriate to represent it with an attribute of another concept which it refers to*
  Example: *city* will be represented as an attribute because, in addition to the name, there is no other interest in it

Representation issues

- *If two entities have been identified (or more) and in the requirements appears a concept that associates them, this concept can be represented by a relationship.*
  Example: The concept of *participation to a course* is representable as a relationship between *participants* and *courses*
- *If one or more concepts turn out to be special cases of another, it is appropriate to represent them by means of a generalization.*
  Example: *professional* and *employees* → *participant*.

Design Patterns

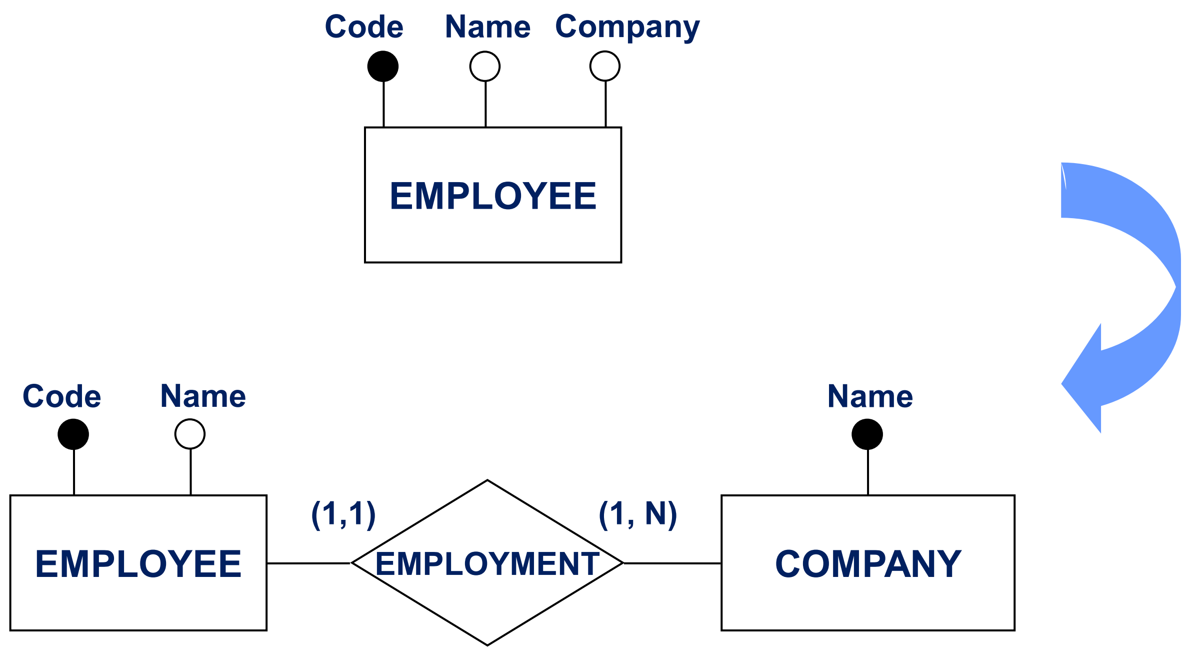
The conceptual representation of data is facilitated by the use of design patterns, that is, by design solutions to common problems.

The design patterns are widely used in software engineering.

Let's see some common patterns in the conceptual design of databases.

Reification of entity attribute

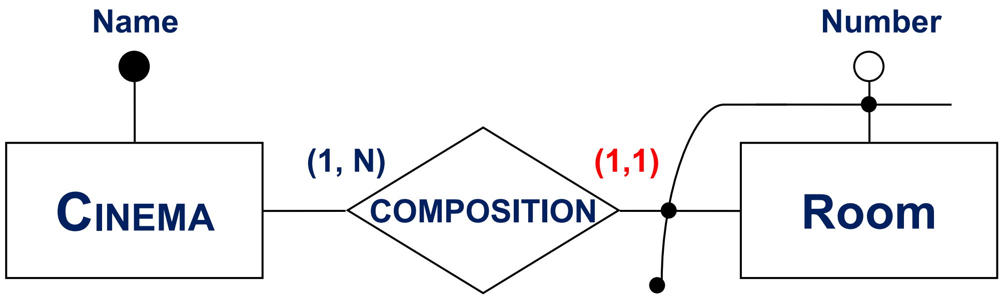
It is a design pattern that involves an entity: it is used when a property deserves to be explicitly represented as a concept (entity).



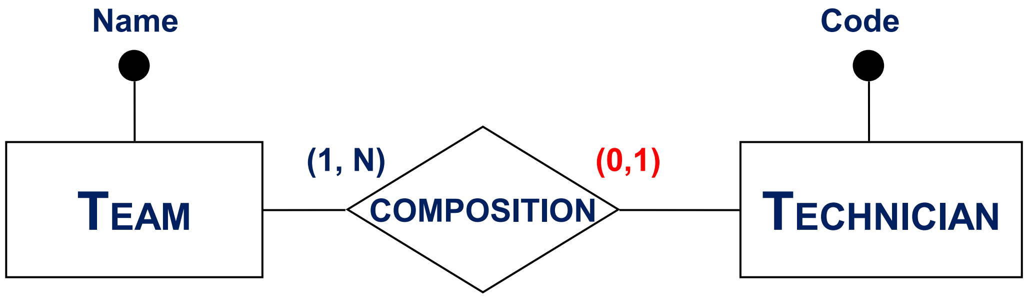
Part-of

It is a pattern that applies to relationships. It is used when an entity is part of another entity. There are two possible forms, depending on whether an entity occurrence ("part“) depends on the existence of an entity occurrence that contains it, or not.

Composition

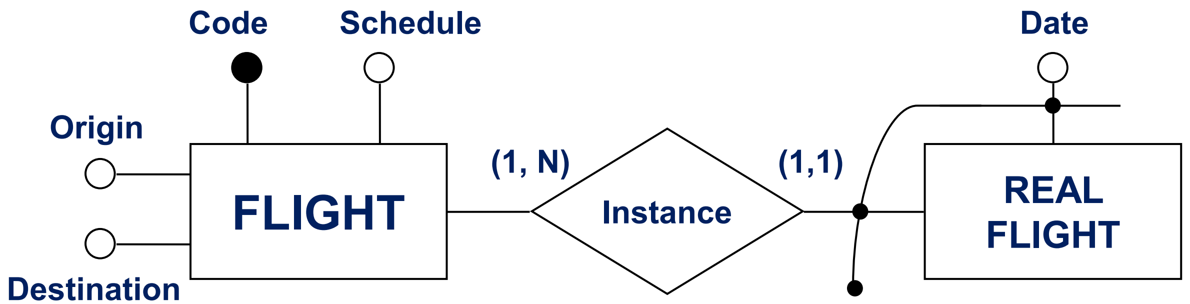


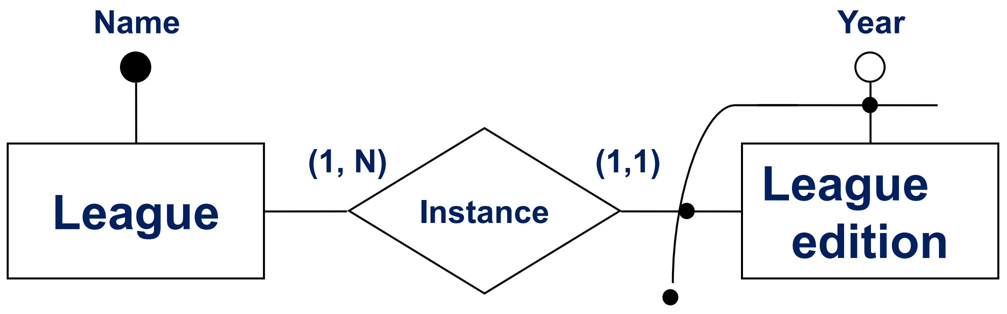
Aggregation



Instance-of

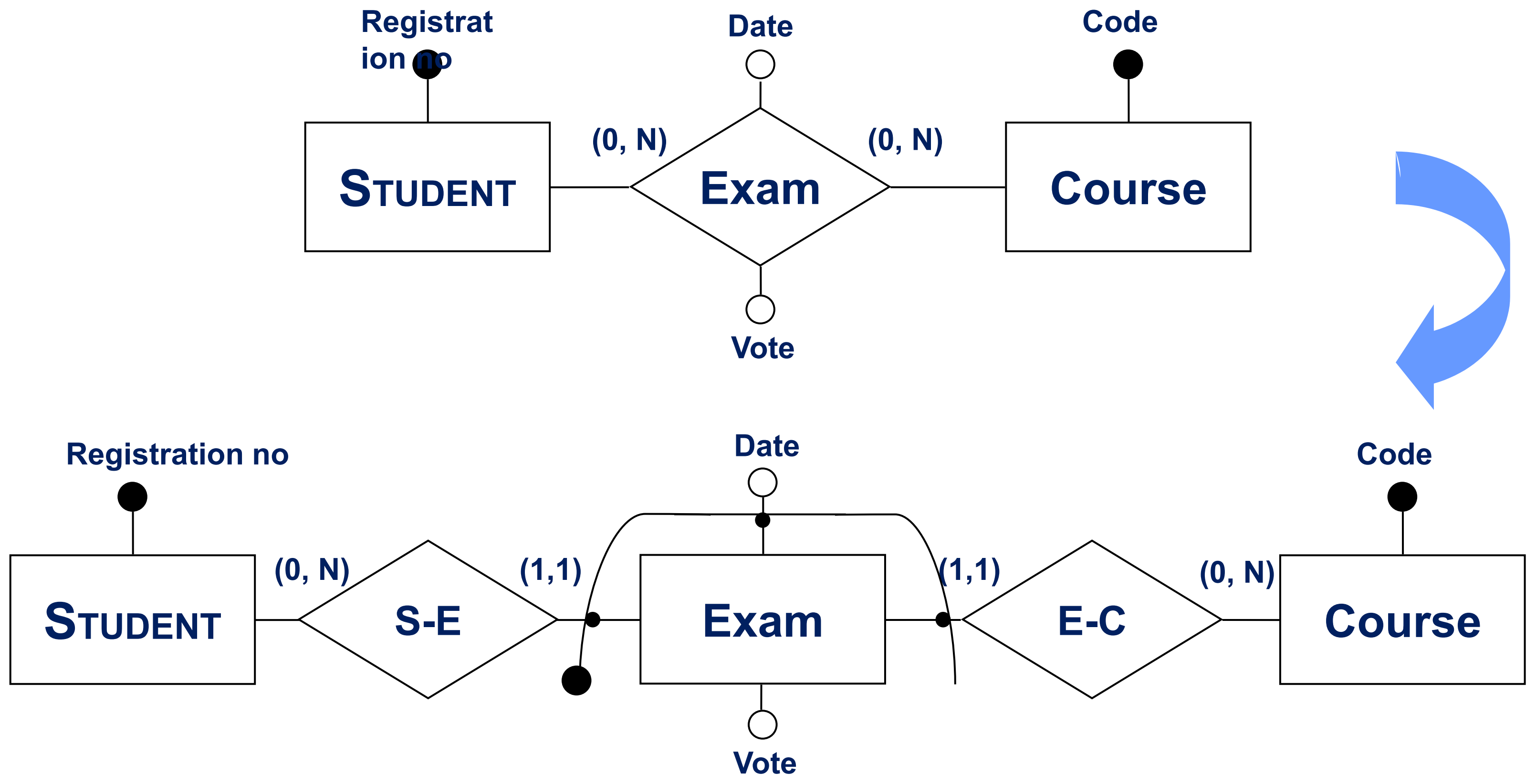
It applies to relationships. It is used when occurrences of an entity of the relationship are instances of the other entity occurrences.





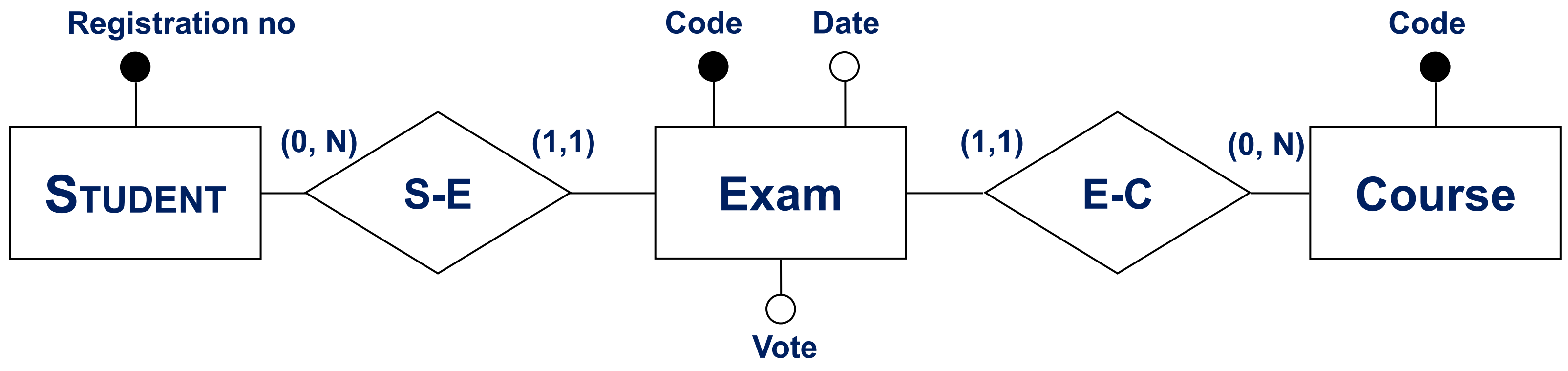
Reification of binary relationships

It applies to represent multiple occurrences of instances of the two associated entities.



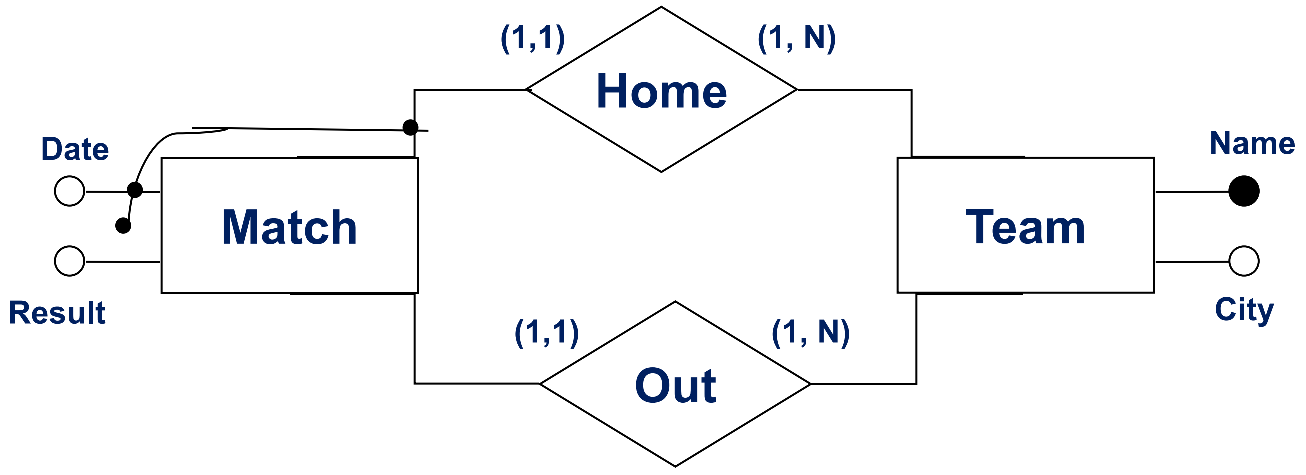
Reification of binary relationships

Alternatively, for a complex external identification, one can introduce an internal identifier for the reified entity.



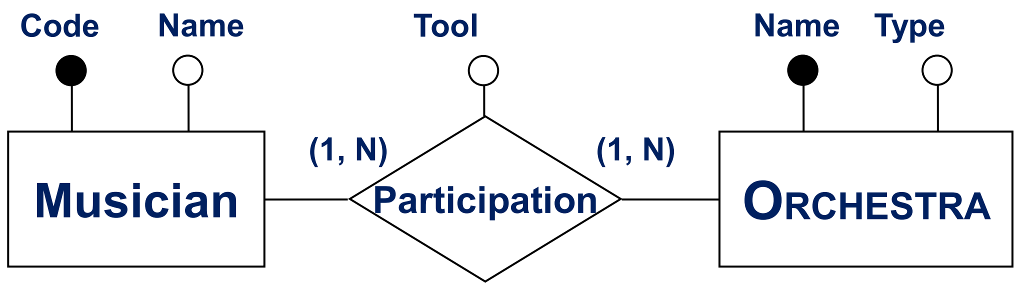
Reification of recursive relationships

It applies to represent multiple occurrences of instances of a recursive relationship.



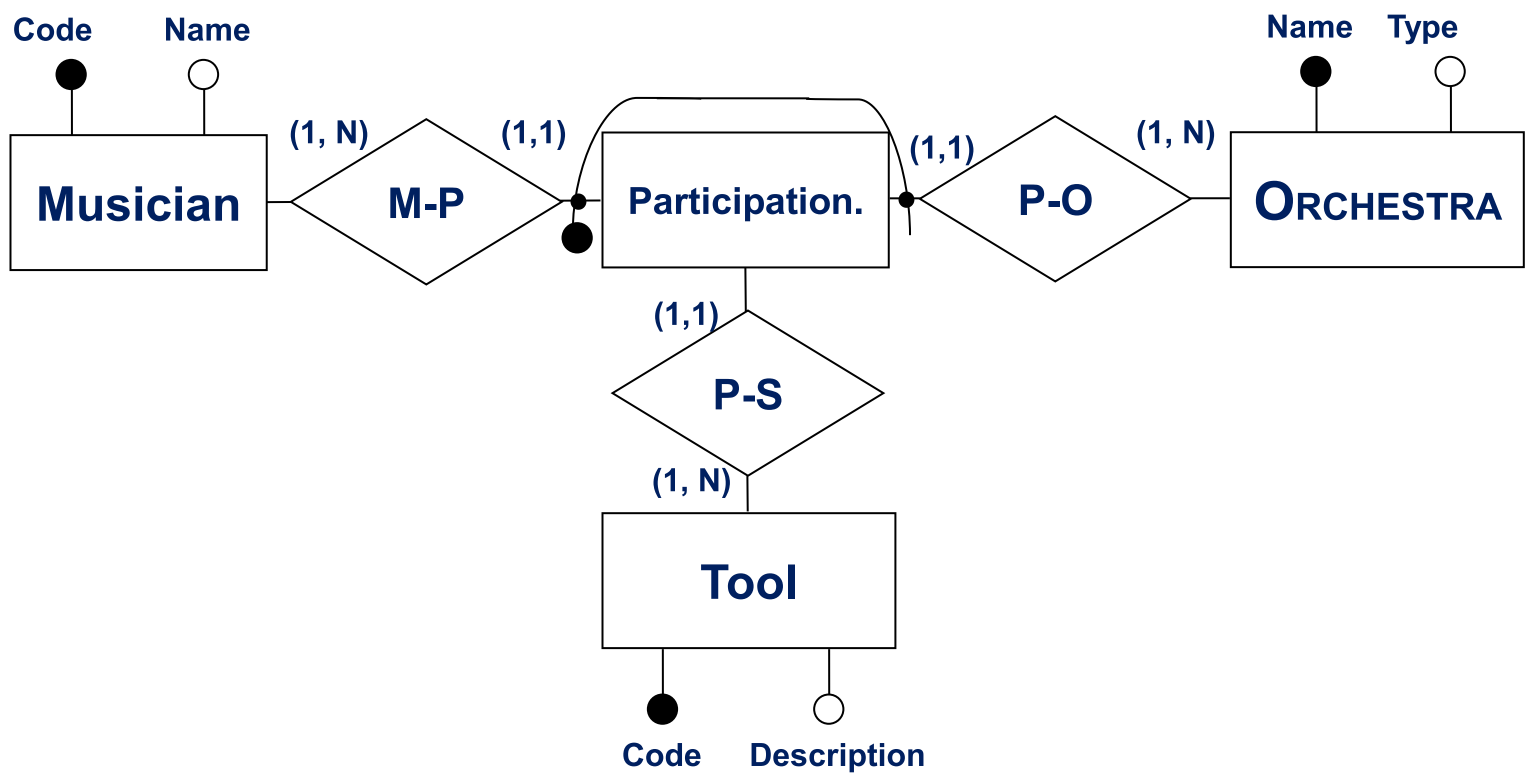
Reification of a relationship attribute

Applies when the relationship property deserves to be explicitly represented as a concept.



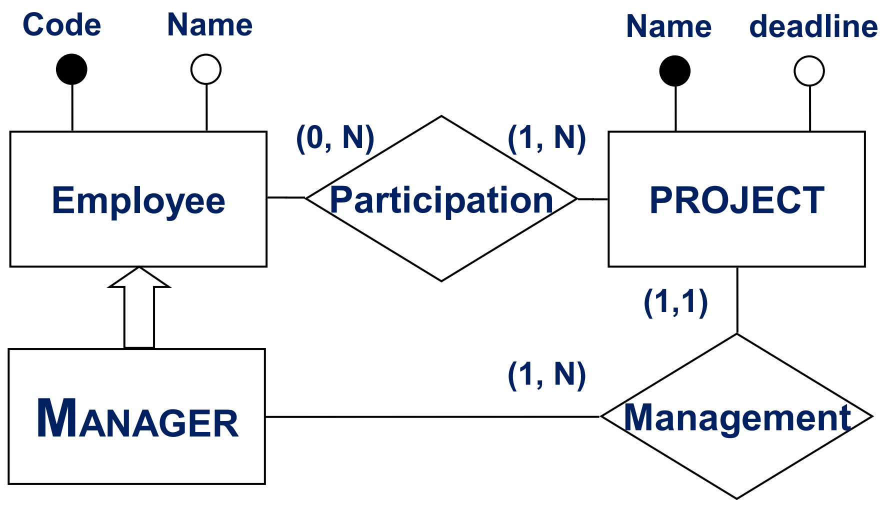
Reification of a relationship attribute

To represent the instrument as an entity we must also reify the relationship.



Pattern for generalizations

It applies when we want to represent a special case of another.

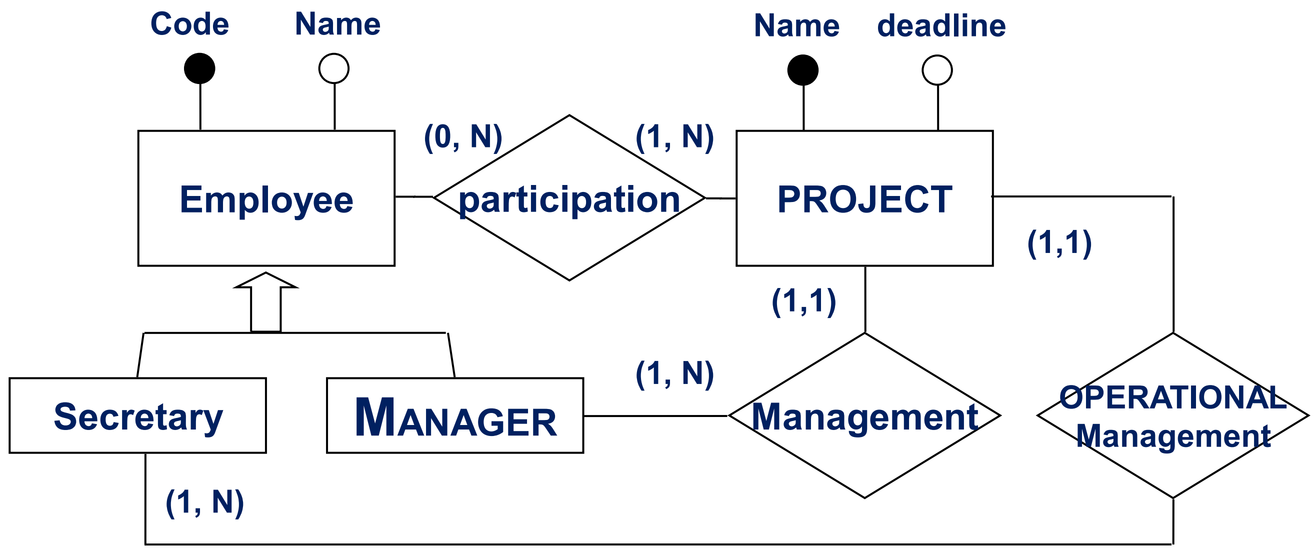


We expect that each pair (managers, project) that appears between the occurrences of *Management* also appears between the occurrences of *Participation*. This constraint is specified by a rule added to the documentation.

Pattern for generalizations

The previous pattern can be generalized to the case where there are more special cases to consider.

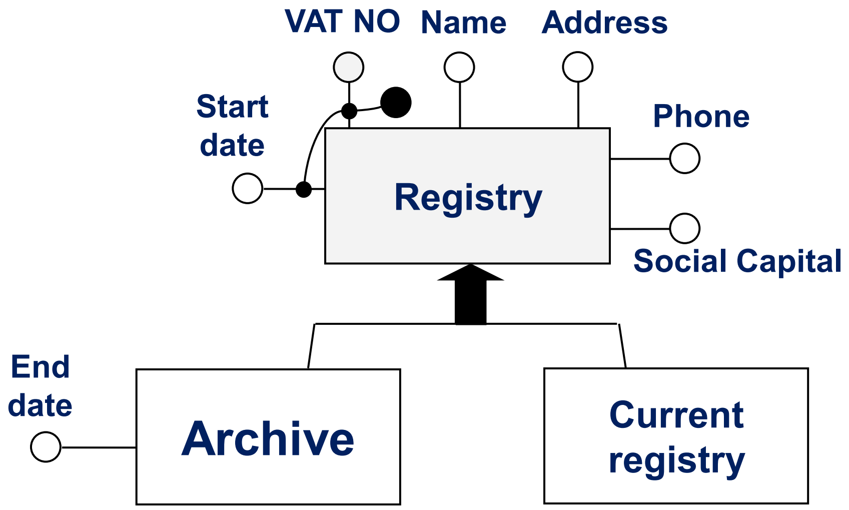
The generalization entity with more children and relationships should be distributed, depending on the specificity of the various entities participating in the generalization.



Archiving a concept

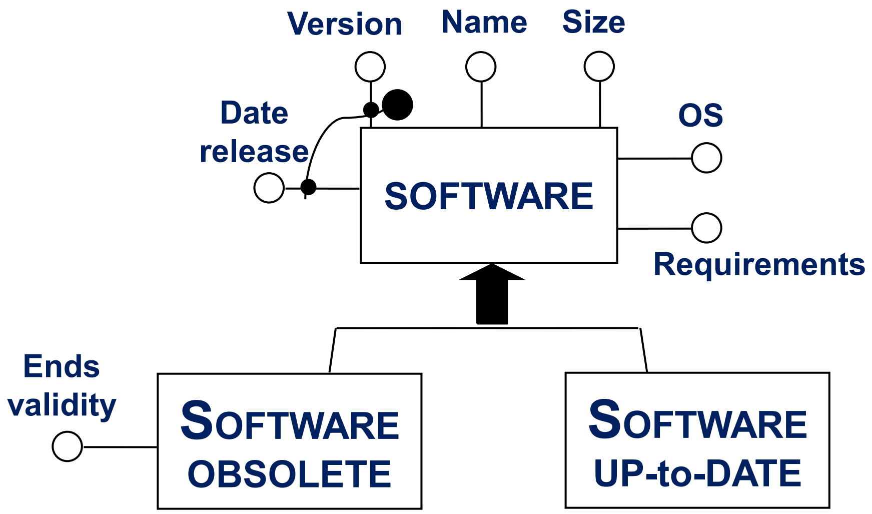
Other design patterns concern archiving of a concept.

If the concept is an entity, we can specialize it into two entities with the same attributes: one is the concept of interest with the updated information, the other the "historic" one. For the historic, we must also represent the end date of the time interval of validity.



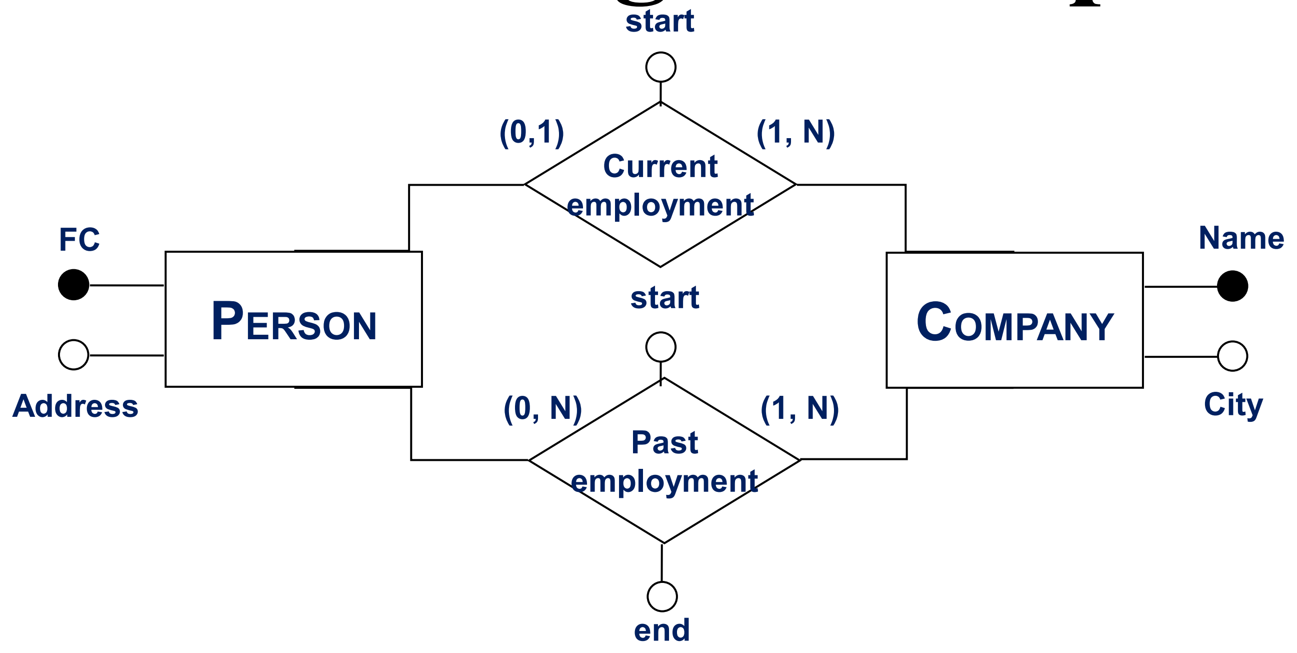
Archiving a concept

Another example:



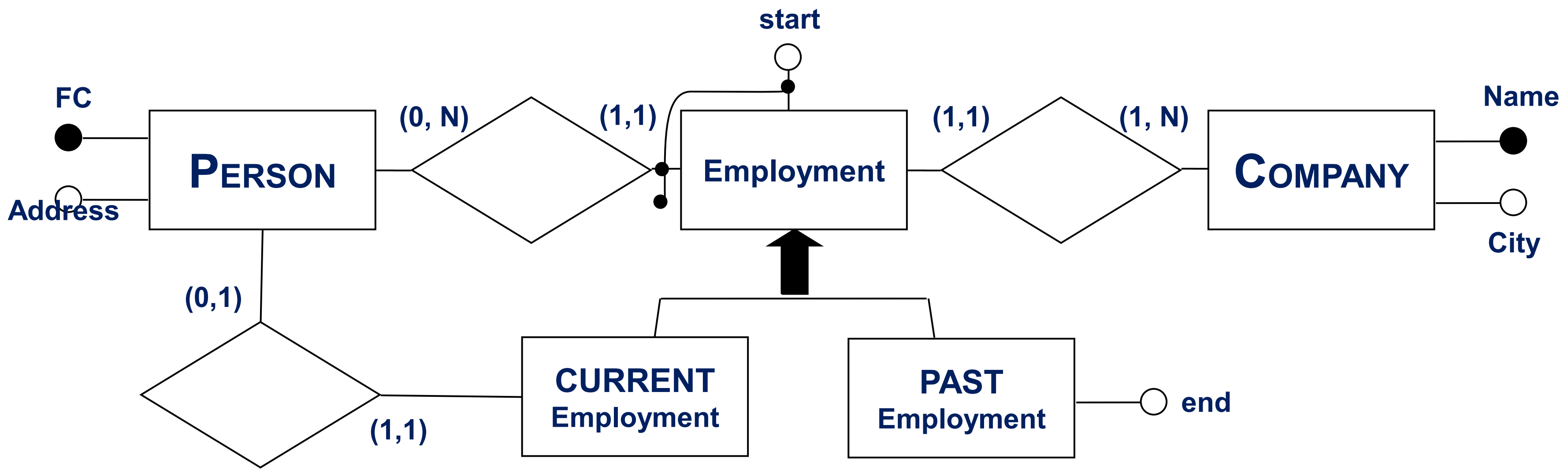
The archiving of a relationship consists in the separation of the representation of the current data from the historical ones by means of two distinct entities.

Archiving a concept



We can not represent the fact that a person has previously worked for the same company at different times. To solve the problem we need to reify the relationship *Employment*

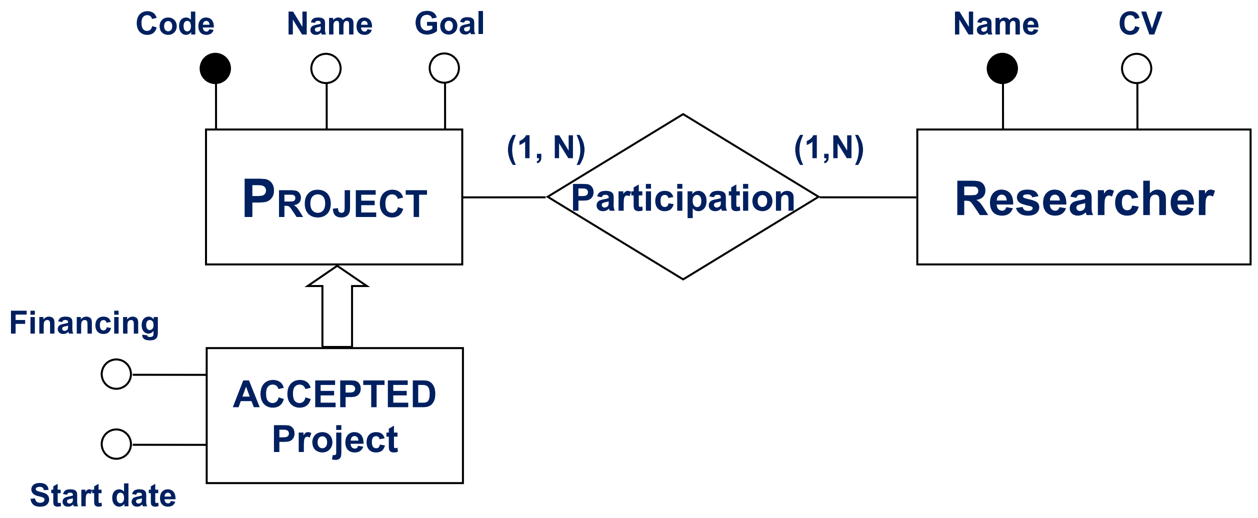
Archiving a concept



We need to add an external constraint to the scheme: all occurrences of the association between *Person* and *current employment* must also appear between the occurrences of the relationship between *Person* and *Employment*.

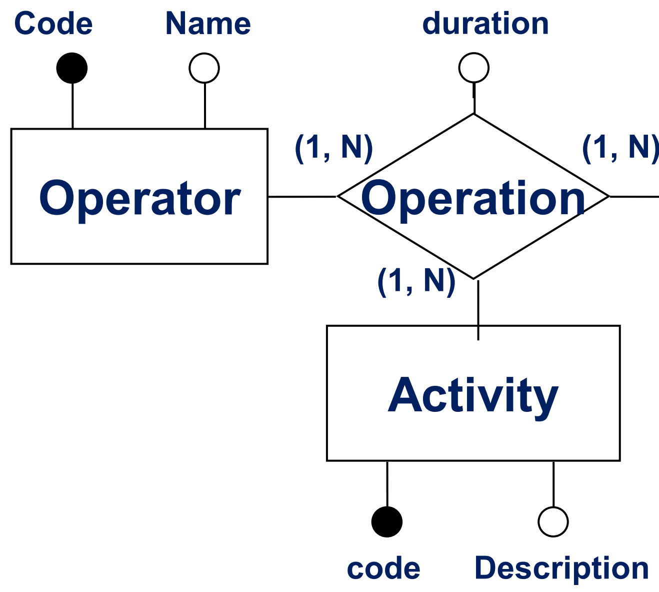
Concept Evolution

Sometimes we want to represent the fact that a concept evolves over time. If the concept is an entity, we can use the generalization to represent the evolution, given that the evolution adds new properties (attributes).



Ternary Association

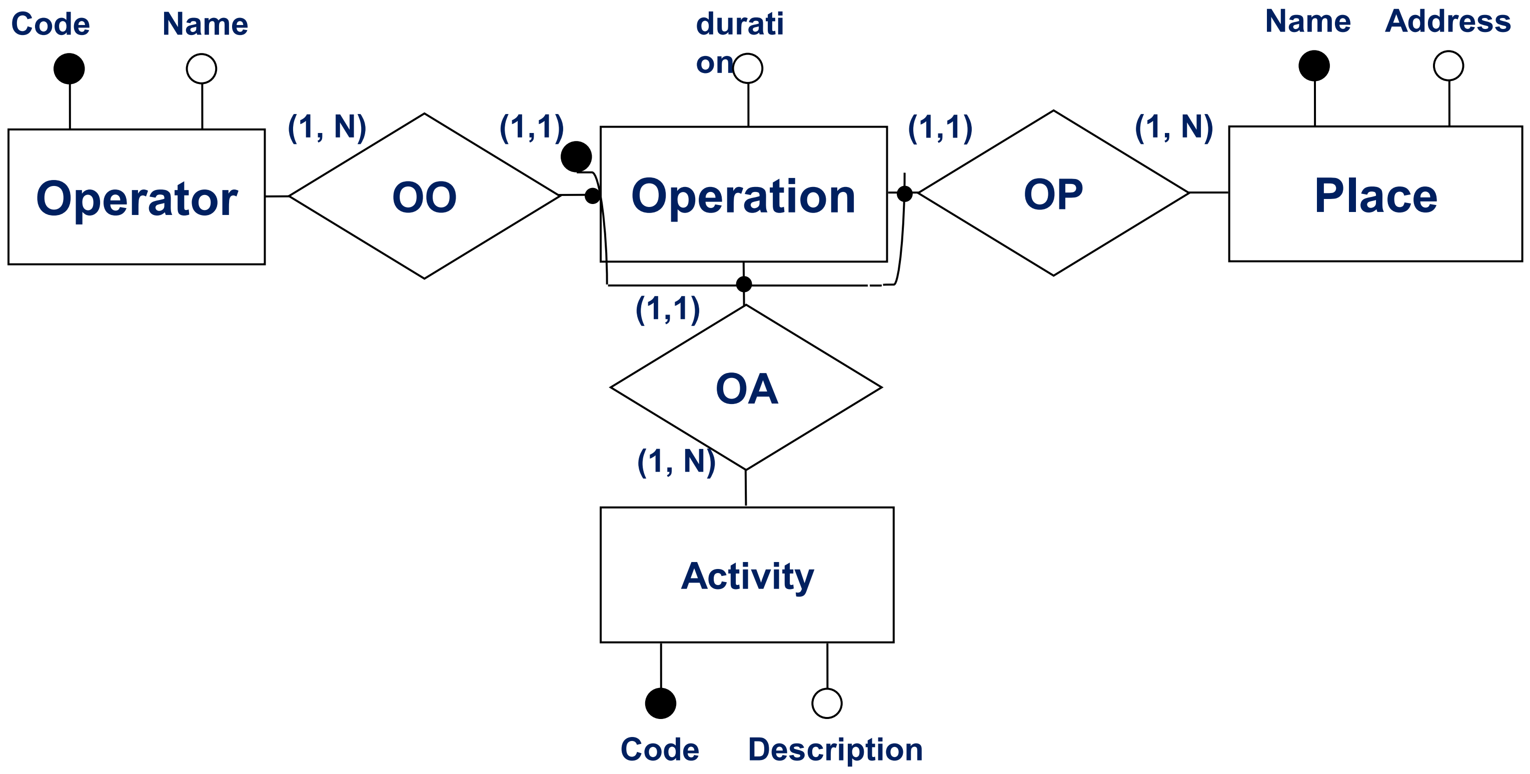
Another design pattern is that of the n-ary associations.



Relationships involving more than 3 entities are generally not recommended, because they seek to represent, with a single construct, concepts independent of each other. This is clear when you apply normalization techniques to conceptual projects.

Reification of ternary relationships

In general it is convenient to reify the *n*-ary relationship.

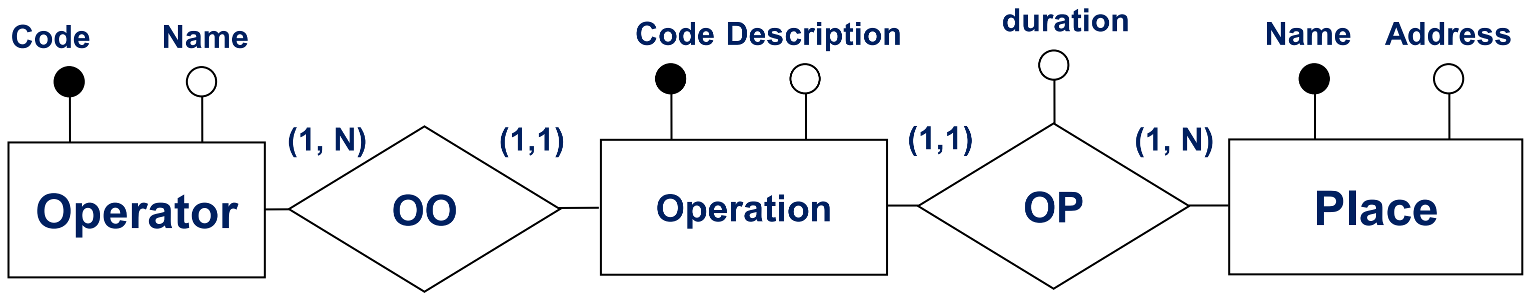


By appropriately changing the identification we can use this pattern even in cases in which the n-ary relation is not good.

Reification of ternary relationships

For example, if at each location, each operator always performs the same activity, an entity *Operation* would be identified only by entities *Place* and *Operator*.

In the extreme case where each activity is carried out in a single location by a single operator, we may use the entity *Activity* as labeling. The diagram is simplified, because the link between activity and place can be represented separately from that between the activity and the operator.

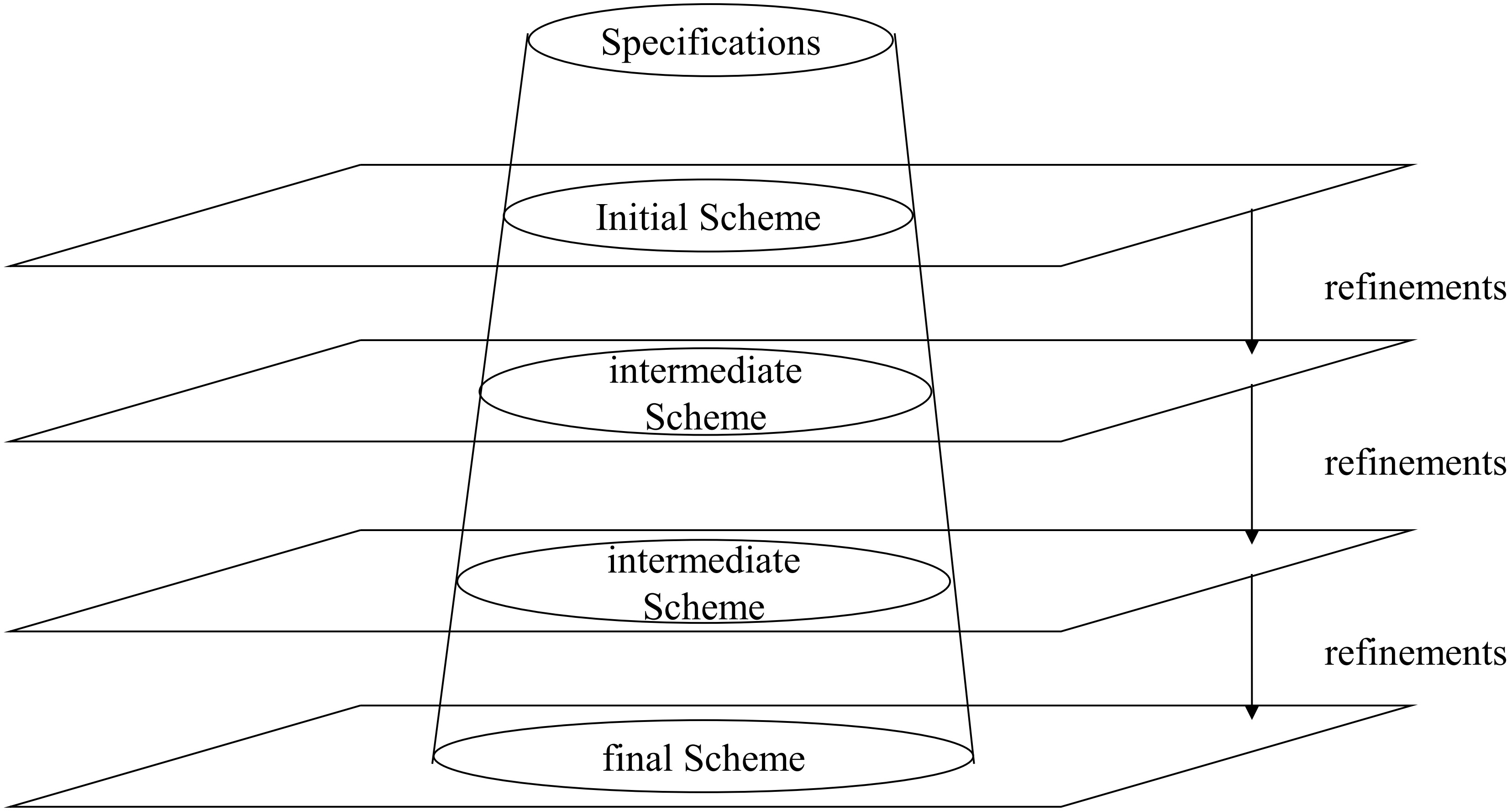


Design strategy: Top-down

With this design strategy, the conceptual framework is obtained as a series of successive refinements from an initial outline that describes all of the specifications with a few very abstract concepts.

The refinements, or ***transformations***, increase the detail of the various present concepts.

Design strategy: Top-down

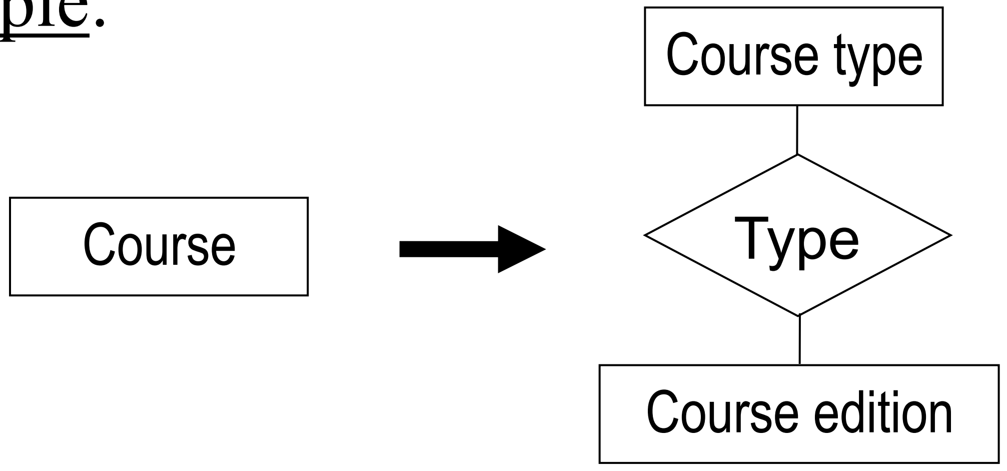


Each of the steps contains a diagram that describes the same information at a different level of detail.

Elementary transformations

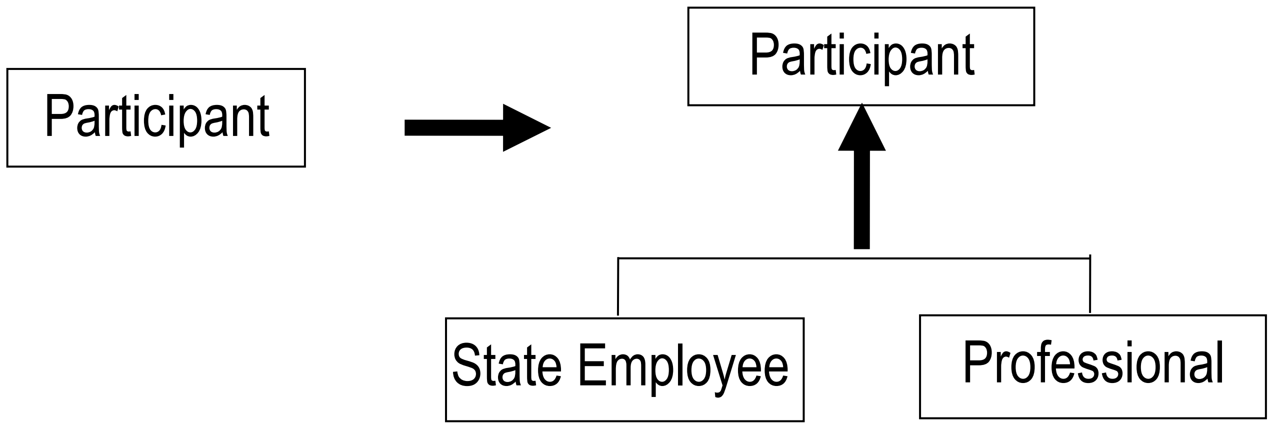
- *From entity to relationship between entities*. It applies when an entity actually describes two different concepts logically related to one another.

  Example:



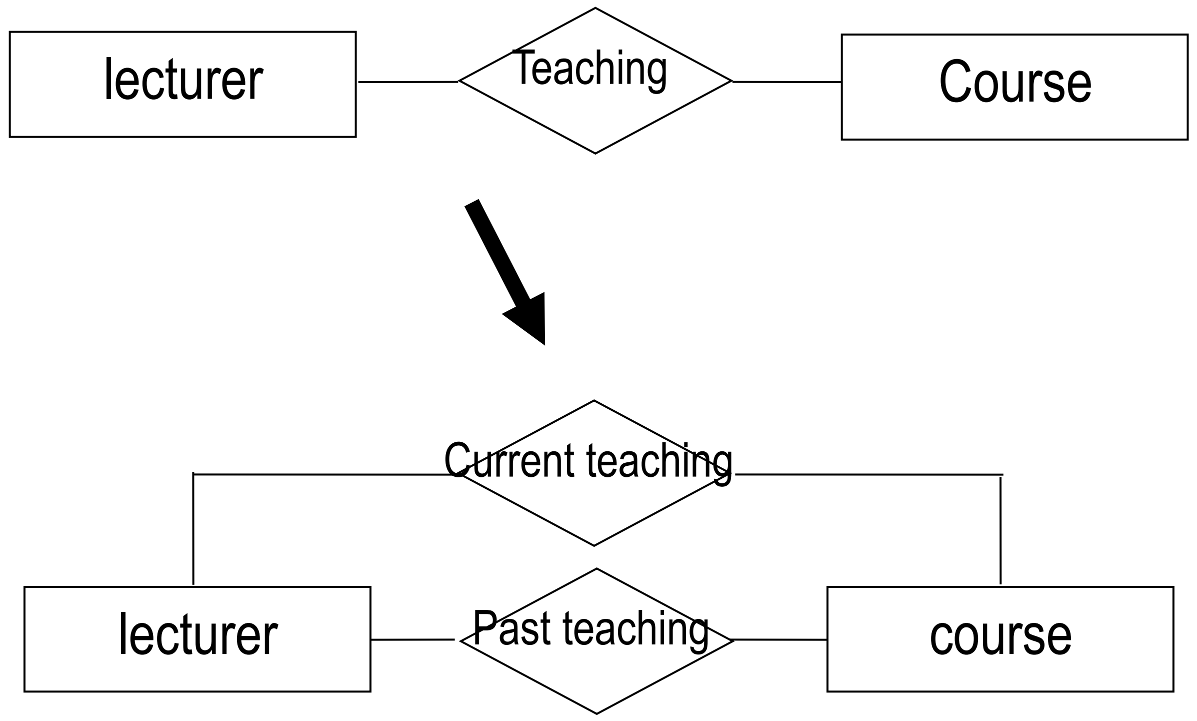
Elementary transformations

From entity to generalization. It applies when an entity is composed of distinct sub-entities.



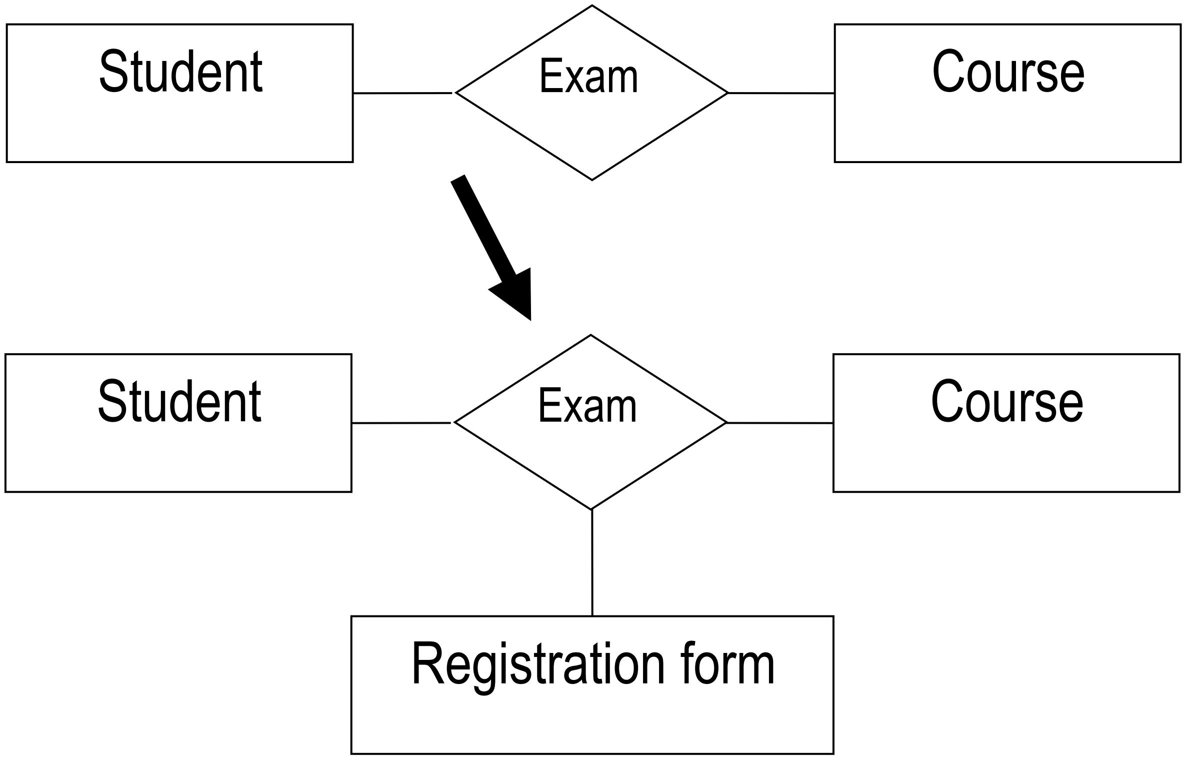
Elementary transformations

- *From relation to set of relationships*. It applies when a relationship describes two different relationships between the same entities.



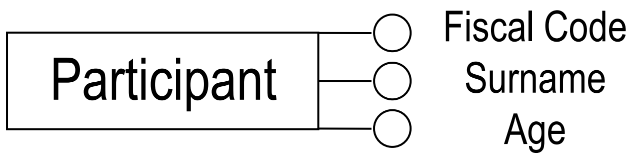
Elementary transformations

- *From relation to entities with relations*. It applies when a relationship actually describes a concept with autonomous existence for the application.

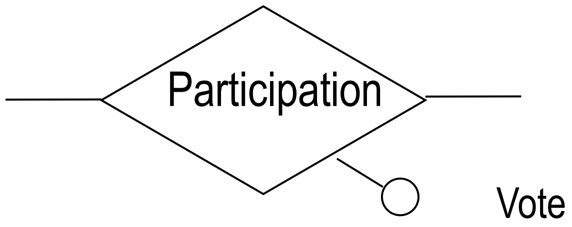


Elementary transformations

- *Introduction of attributes of entities*.



- *Introduction of attributes of relations*.



Elementary transformations

Benefits (The top-down strategy): the designer may initially describe all the specifications of the data by neglecting the details, and then going into the details of a concept at a time.

Downside: We must have a global and abstract view of all components of the computer system.

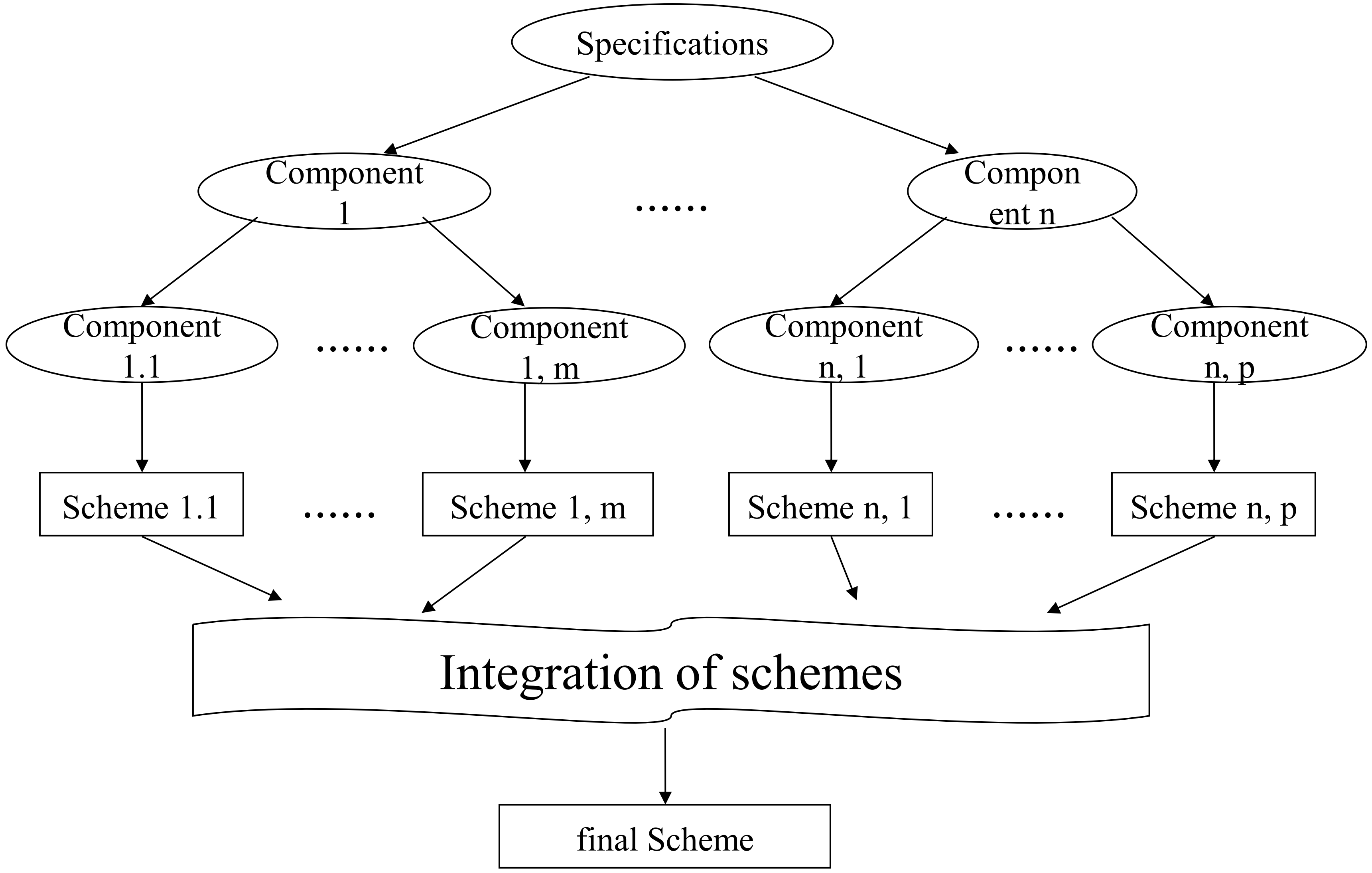
Project strategy: Bottom-up

The initial specifications are divided into progressively smaller and smaller components, until these components describe an elementary fragment of the reality of interest.

At this point, the various components are represented by simple conceptual schemes which can also consist of individual concepts.

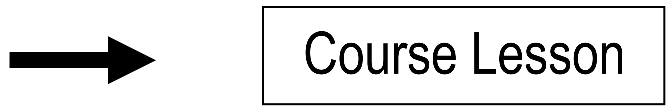
The various schemes are then fused up to reach, through a complete integration of all components, the final conceptual schema.

Project strategy: Bottom-up



Bottom-up: Primitive transformations

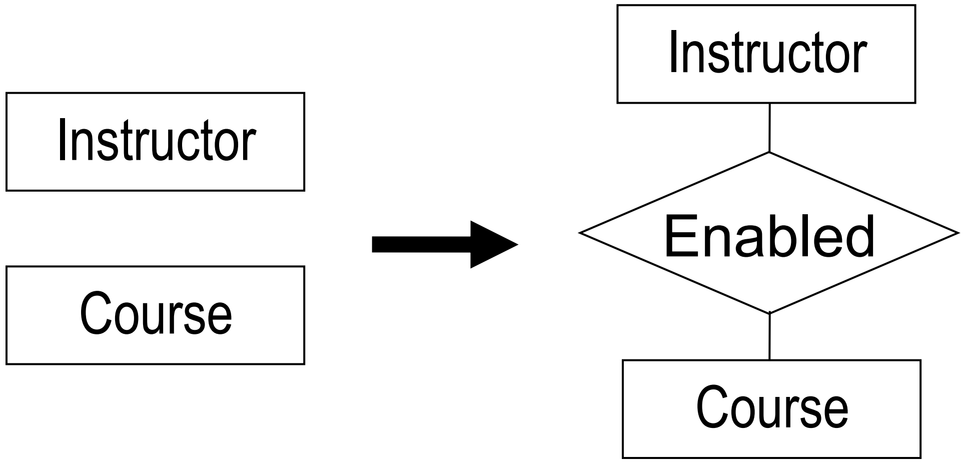
- *Entity Generation*. It applies when you locate in the specifications a class of objects with common properties.
  Example:



Bottom-up: Primitive transformations

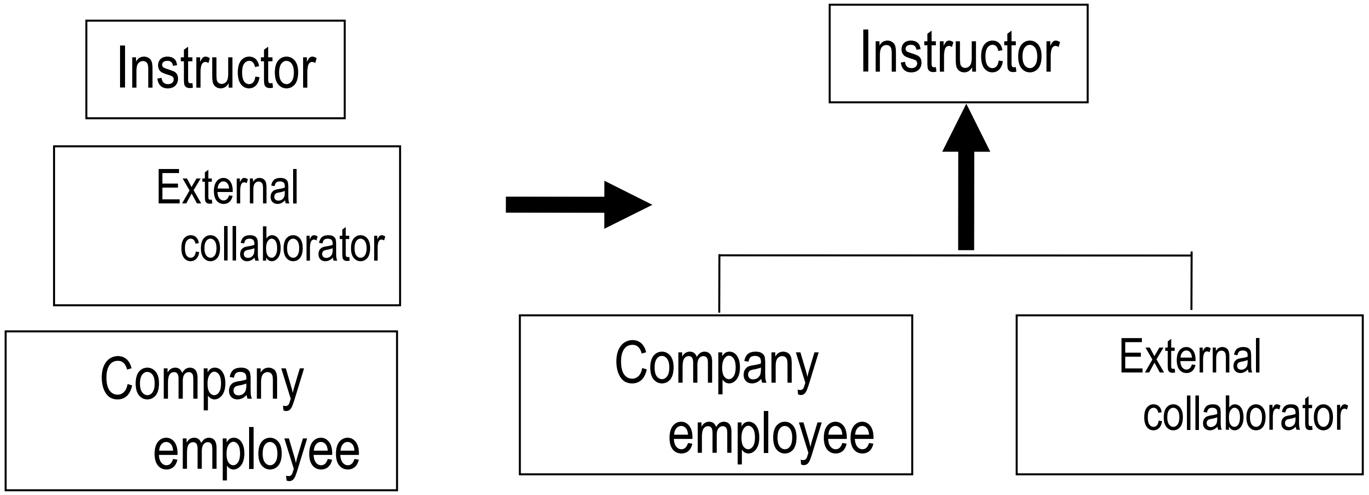
- *Generating relationships*. It applies when you locate in the specifications a logical link between the two entities.

Example:



Bottom-up: Primitive transformations

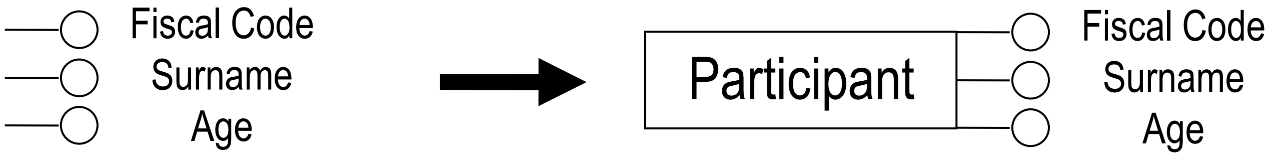
Generation of generalization: It applies when we locate in the specification a link between various entities due to a generalization.



Bottom-up: Primitive transformations

- *Aggregation of attributes in entities*: It applies when, starting from a series of attributes, it is possible to identify an entity that can be seen as an aggregation of these attributes.

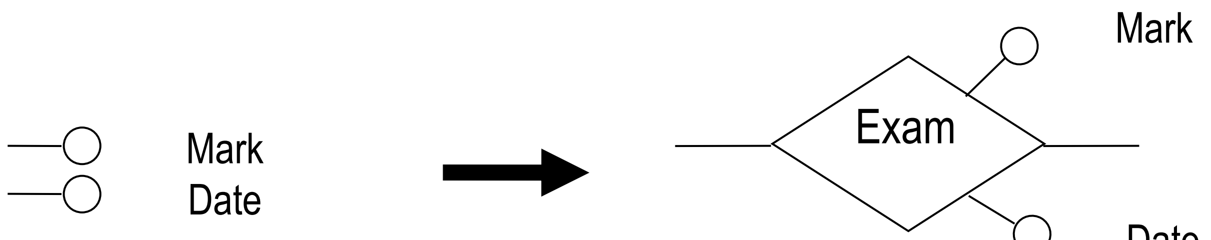
  Example:



Bottom-up: Primitive transformations

- *Aggregation of attributes in relationship*. It applies when we locate a relationship that can be viewed as an aggregation of some attributes.

  Example:



Transformation bottom-up

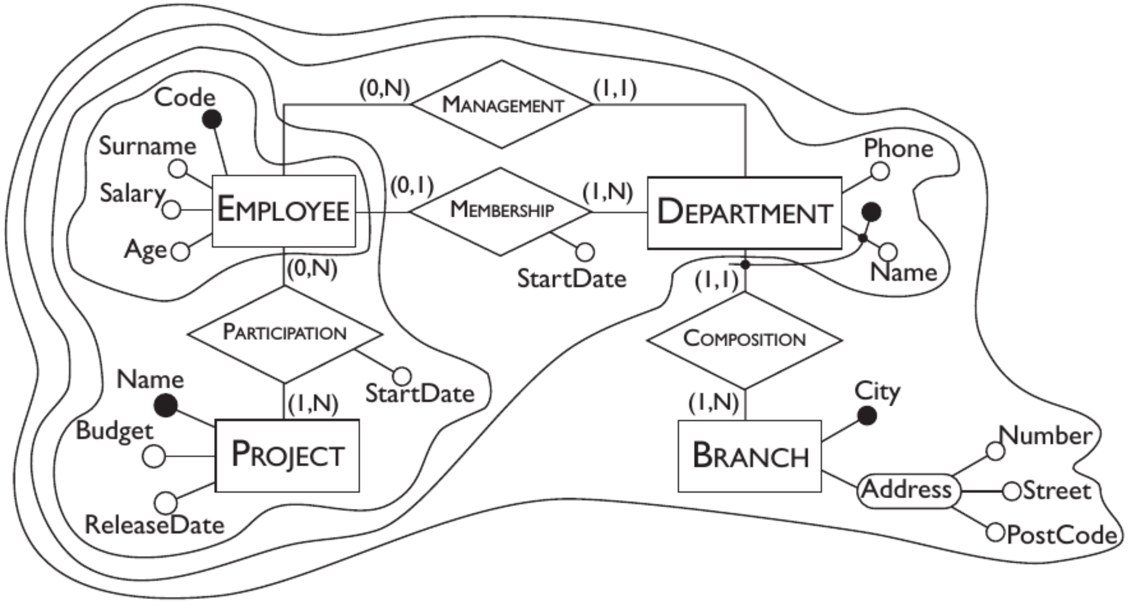
Benefit: Adapts well to work carried out in collaboration or divided within the group.

Downside: It requires the integration of operations of different conceptual schemes that, in the case of complex schemes, almost always present major difficulties.

Project strategy: Inside-Out

It is a special case of bottom-up strategy.

They initially identify some important concepts and then proceed, from these, in "wildfire" way, representing first the concepts conceptually closer to the initial concepts and then move to the more distant ones.



Project strategy: Inside-Out

Benefit: Do not require steps of integration.

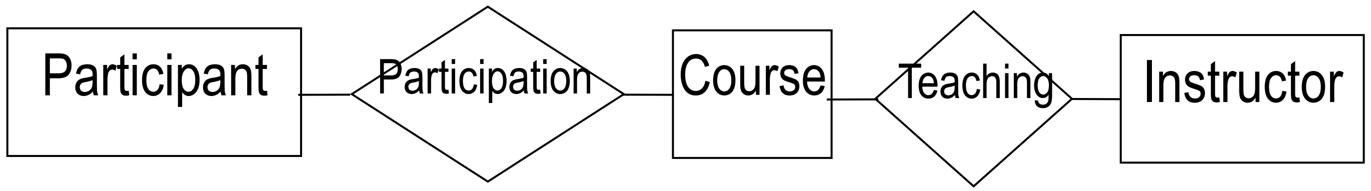
Downside: Lacks of abstraction. It is necessary, from time to time, examine all the specifications to find not yet represented concepts and describe the new concepts in detail.

Project strategy: Hybrid

It is the design strategy that is usually followed in real cases (complex).

They are divided requirements into separate components (bottom-up) but at the same time identifies the general and fundamental concepts (***skeleton schema***).

Example:



Project strategy: Hybrid

Then, it is refined, expanded, aggregates.

This strategy is the most flexible and adapts well to opposing needs: to first divide a complex problem into sub-problems and then to proceed by successive refinements.

Moreover, the documentation may still be organized in a top-down manner.

Often we proceed by sectors and then by "integrating".

The skeleton schema definition: "Initial Design"

- The most important concepts are identified,

for example, because most cited or because explicitly indicated as crucial,

- and arranges them into a simple conceptual scheme;

it is important to focus on the essentials: many attributes, cardinality, structured hierarchies can be postponed.

A general methodology

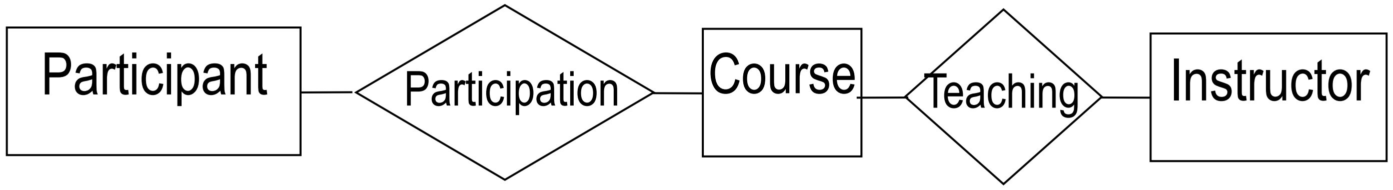
- ***Requirements analysis***.
  - Analyzing the requirements and eliminating the ambiguities present;
  - Grouping requirements in homogeneous groups
  - Build a glossary of terms;
- ***Basic Step***.
  - Identify the most relevant concepts and represent them in a skeleton outline.
- ***Decomposition Step*** (Where appropriate and necessary).
  - Make a decomposition of the requirements with reference to the concepts present in the skeleton scheme.

A general methodology

- ***Iterative Step***: To be repeated, for all the sub-schemes (if present), until every specification has been represented.
  1. Refine the concepts based on their specifications.
  2. Add new schema concepts to describe specifications not yet described.
- ***Integration Step*** (To be performed if there are different sub-schemes).
  - Integrating the various sub-schemes in a general diagram referring to the skeleton diagram.

An example of conceptual design

We have already carried out the analysis of the requirements and we have already built the skeleton shema.



With reference to this scheme we can decide to separately analyze the specifications for the *participants*, the *courses* and *teachers* and proceed to wildfire.

Requirements analysis

Phrases on participants

For participants (5000), identified by a code, we want to store the tax code, last name, age, sex, city of birth, the name of the company they work now and the companies for which worked in the past (with the period of work), editions of the courses they have attended, with its final mark in tenths, and the period of attendance. We also represent the editions of courses they are currently attending.

Phrases for specific types of participants

For participants who are professionals, we represent the area of interest and, if they have, the professional title. For participants who are state employees, we represent their level and position held.

Phrases related to the company

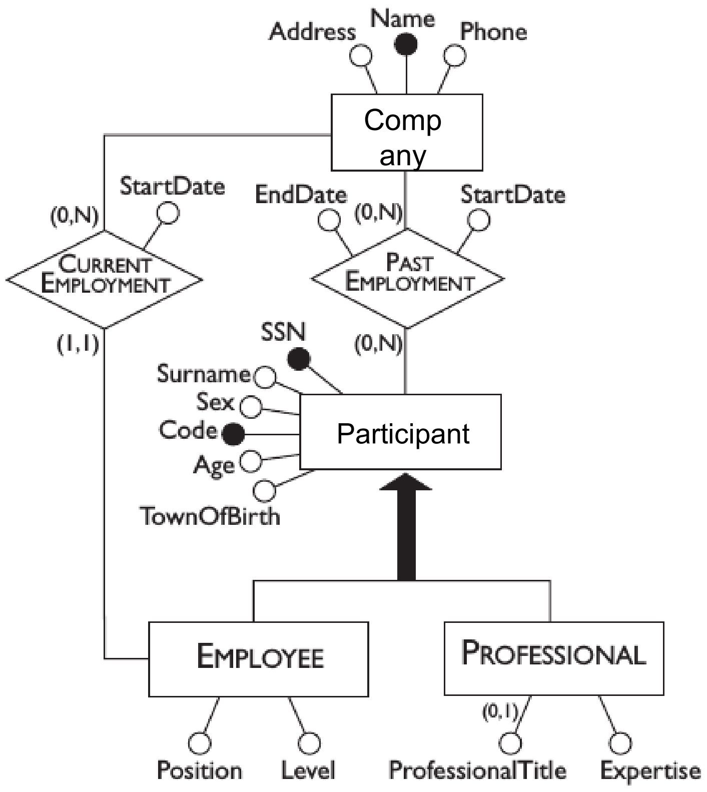
As regards the companies for which participants are currently working and / or have worked, we represent the name, address and telephone number.

An example of conceptual design

Iterative Step. We identify three types of participants: *professionals*, *state employee* and "all others" (*private employees*). These entities are children entities of *participant*: the generalization is total. An entity should be introduced now to represent the company where the employees work. The relationship between company and the private employee can be divided into two relationships: *past employment* and *current employment*, with different properties.

An example of conceptual design

Adding attributes to entities and relationships, cardinality relations and identifiers to the entities, you get the following scheme:



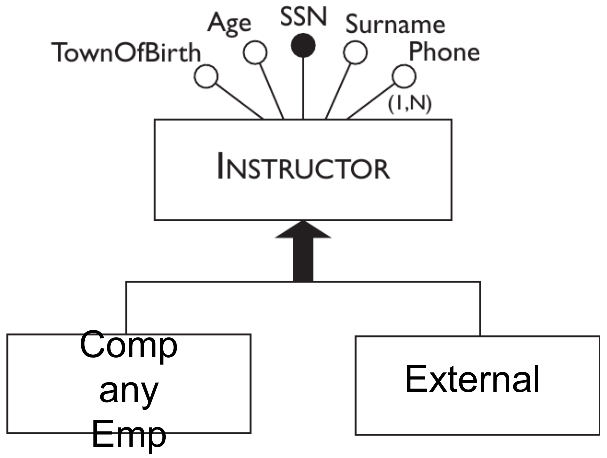
Requirements analysis

Phrases related to instructors

For instructors (about 300), we represent the name, age, town of birth, all the phone numbers, the title of the course they teach, those which have taught in the past and those they can teach. Instructors can be company employees or external collaborators.

An example of conceptual design

For instructors, they should only be distinguished employees of the training company by external collaborators. We introduce the fiscal code of the instructor as an identifier.



Requirements analysis

Phrases related to courses

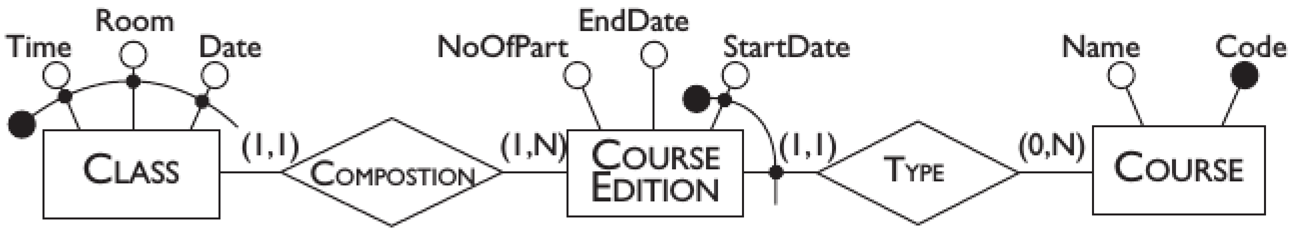
For courses (around 200), we represent the title and code, the various editions and, for running editions, we represent the day of the week, the classrooms and the hours where classes are held and the number of participants.

An example of conceptual design

By analyzing the entity *course*, we distinguish two concepts related to each other but clearly distinct: the abstract concept of *course* (which has a name and a code) from *edition of the course* that has a start date, an end date and a number of participants (design pattern of istance-of). We represent these two distinct entities concepts related by *type*.

We have also to represent the *classes* which we can describe as an entity linked to the editions of courses from a relationship called *composition*.

An example of conceptual design



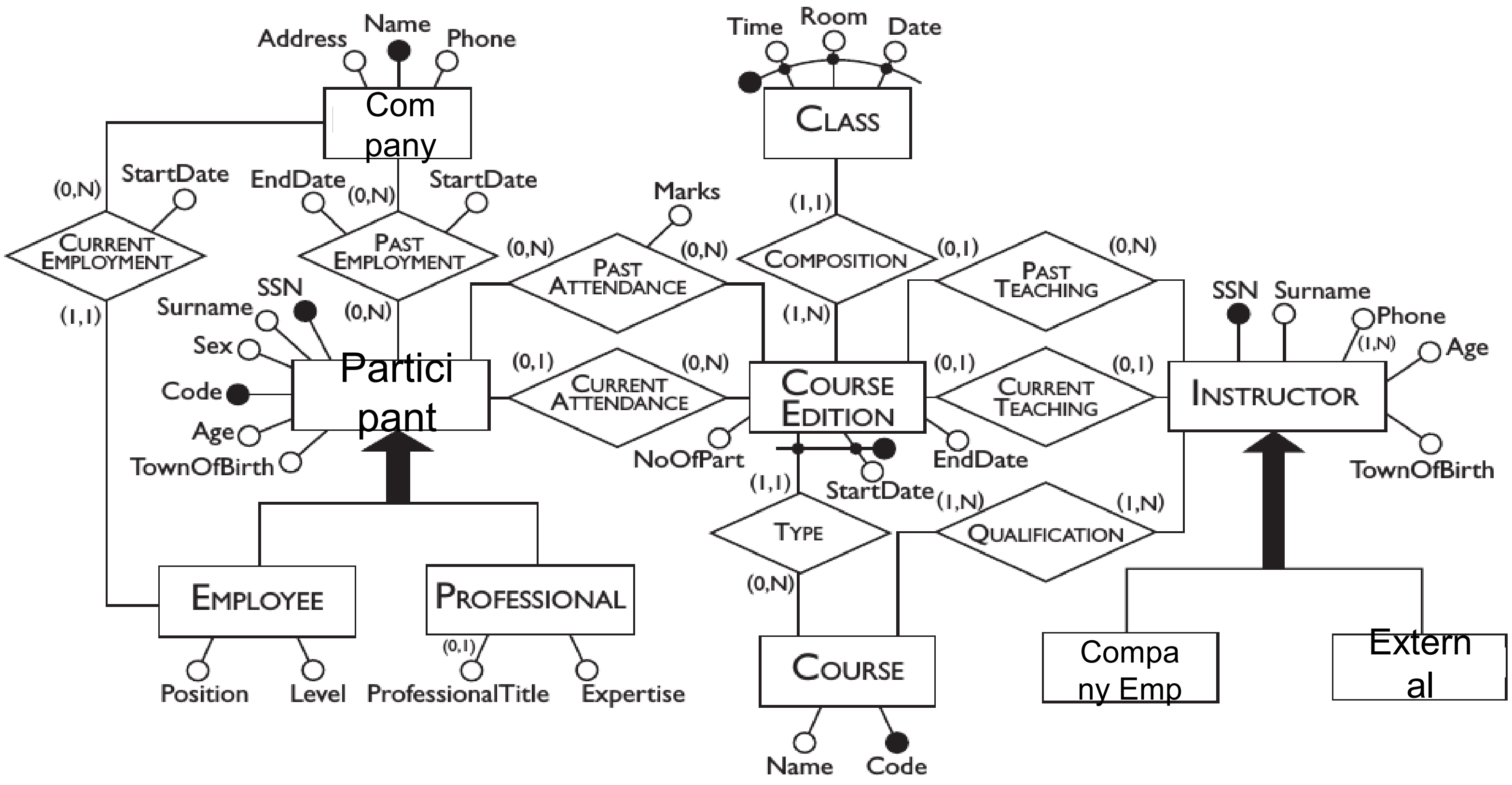
The final schema is obtained by integration of the schemes obtained up to this point.

An example of conceptual design

The connection between the two sub-schemes for instructors and for courses occurs through a teaching relationship that has to be refined. We can identify here three types of correlation: the *current lectures*, those *past lectures* and *enabled to teach*.

The integration of the diagram obtained with that of *participants* passes through the refinement of the relationship *participation* in two relationships: *current* and *past participation*.

An example of conceptual design



The teachers of a course edition can change

An example of conceptual design

There is a redundancy: The attribute Number of participants in the entity Course Edition can be derived, for a certain edition, counting the number of instances of the entity Participant that are related to this edition.

Finally, it must complete the scheme with a suitable *documentation*. In particular, we have to specify any *constraints* not expressed directly in the schema.

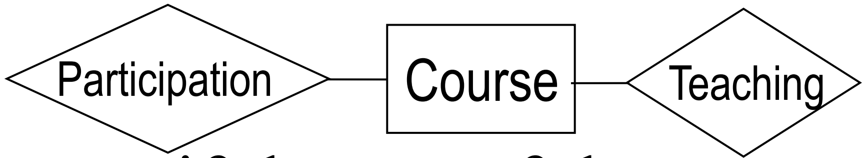
Example A teacher can teach in a course only if it is empowered to do so.

Quality of the conceptual model

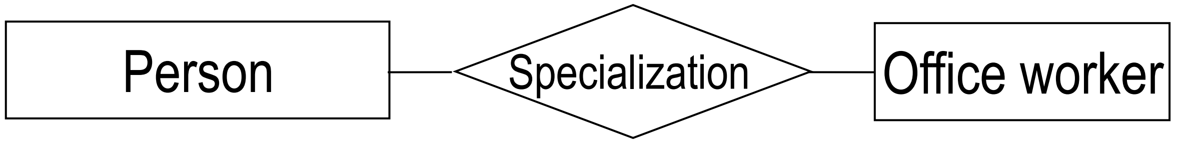
In the construction of a conceptual scheme, it must be assured the quality of the schema property.

Correctness. A conceptual scheme is:

- Syntactically correct when the constructs are used properly.



- Semantically correct, if the use of the constructs does not meet their definition.



This is verified by inspecting the schema and comparing the concepts with the specifications.

Quality of the conceptual model

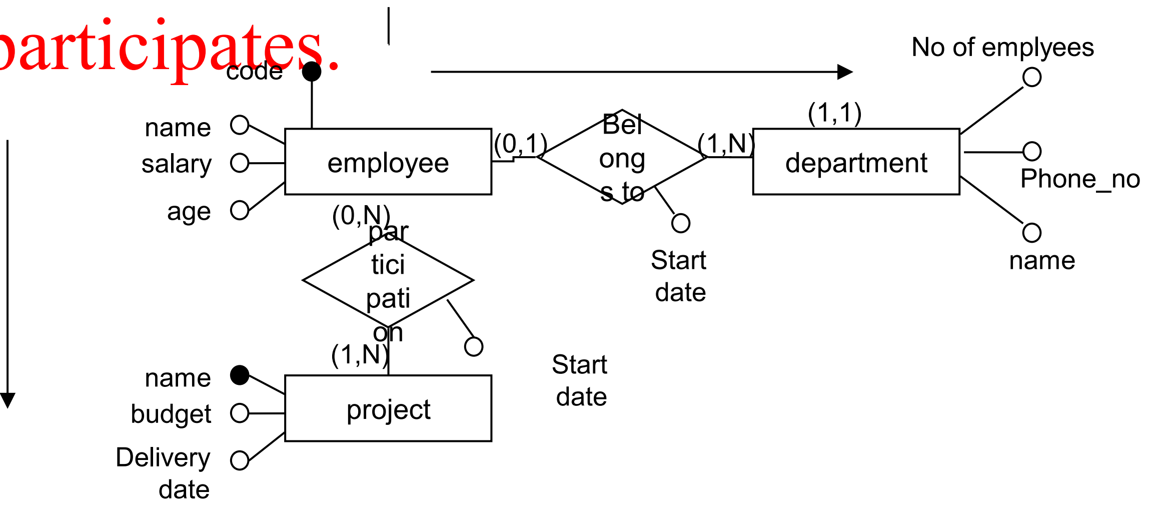
Completeness. All the data of interest are represented and all the operations can be performed starting from the concepts described in the schema.

In contrast to the correctness, it is verified starting from the specifications and checking that the various concepts are shown in the diagram, or that the concepts involved in the operations are reachable "by surfing" through the schema.

Quality of the conceptual model

Example:

Operation: Find the data of an employee, the department in which he/she works and projects in which she/he participates.



Operation: Find projects that are owned by a department. ???

Quality of the conceptual model

Readability. It refers to the cognitive load required to interpret a schema.

Depends on:

- Significance of the names given to the concepts
- Choosing the right level of abstraction (e.g., represent only the most significant attributes of an entity)
- Rendering:
  - Arrange the elements in a grid using as central elements those with multiple links (associations with others).
  - Draw only perpendicular lines
  - Minimizing the intersections
- Arrange the parent entities above children entities.

Quality of the conceptual model

Minimality. A schema is minimal when all specifications are represented only once in the diagram.

However not always the presence of redundancies is indication of poor quality. It affect the efficiency (the maintenance of a redundancy has a cost), which must be evaluated according to the frequency of operations and the data volume.

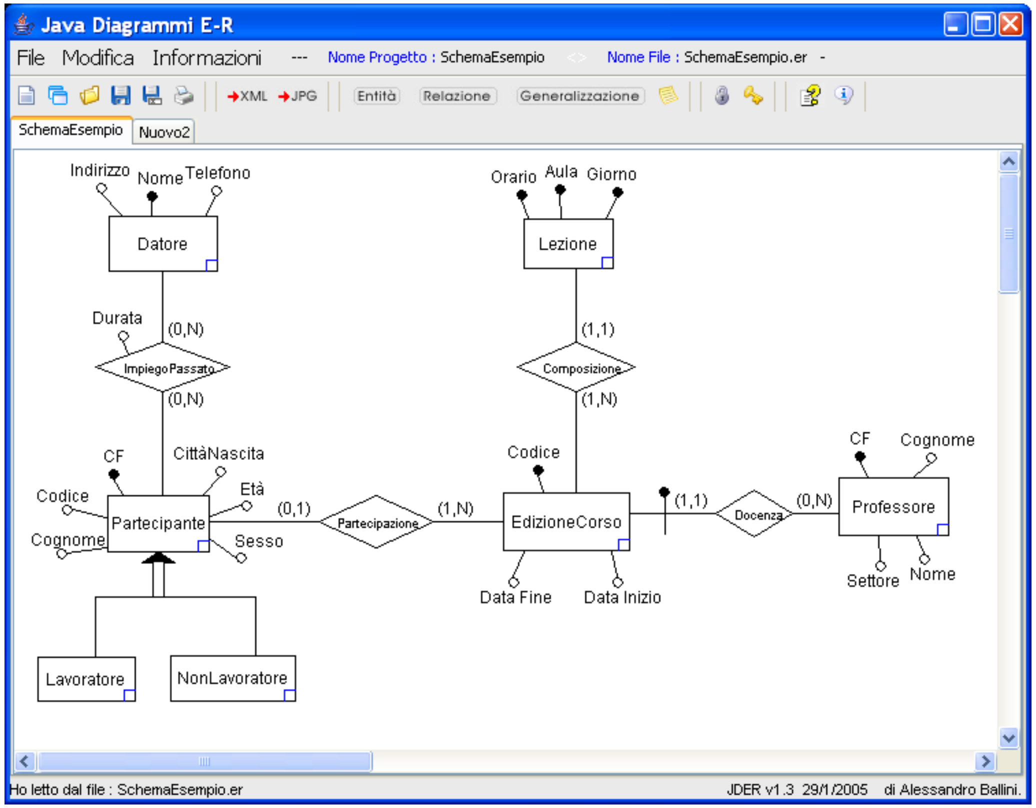
Design Tools

The design activity can be made more productive by making use of

- Simple diagrammers that allow you to graphically represent the elements of a conceptual scheme, but does not allow checks on the semantics nor support the transformation of the conceptual schema in a logical schema;
- Real CASE tools (*Computer Aided Software Engineering*) To aid the engineer of the software and provide support to all major phases of the development of a database (conceptual, logical and physical).

Diagrammers

A simple plotting is JDER (Java ER Diagrams):
http://gianvitopio.wordpress.com/jder/
JDER adopts the notation available on the text of Atzeni *et al*.



Diagrammers

Another plotting (generic) is DIA:
http://live.gnome.org/Dia
DIA adopts different notations, simila to that adopted in the text of Elmasri and Navathe.

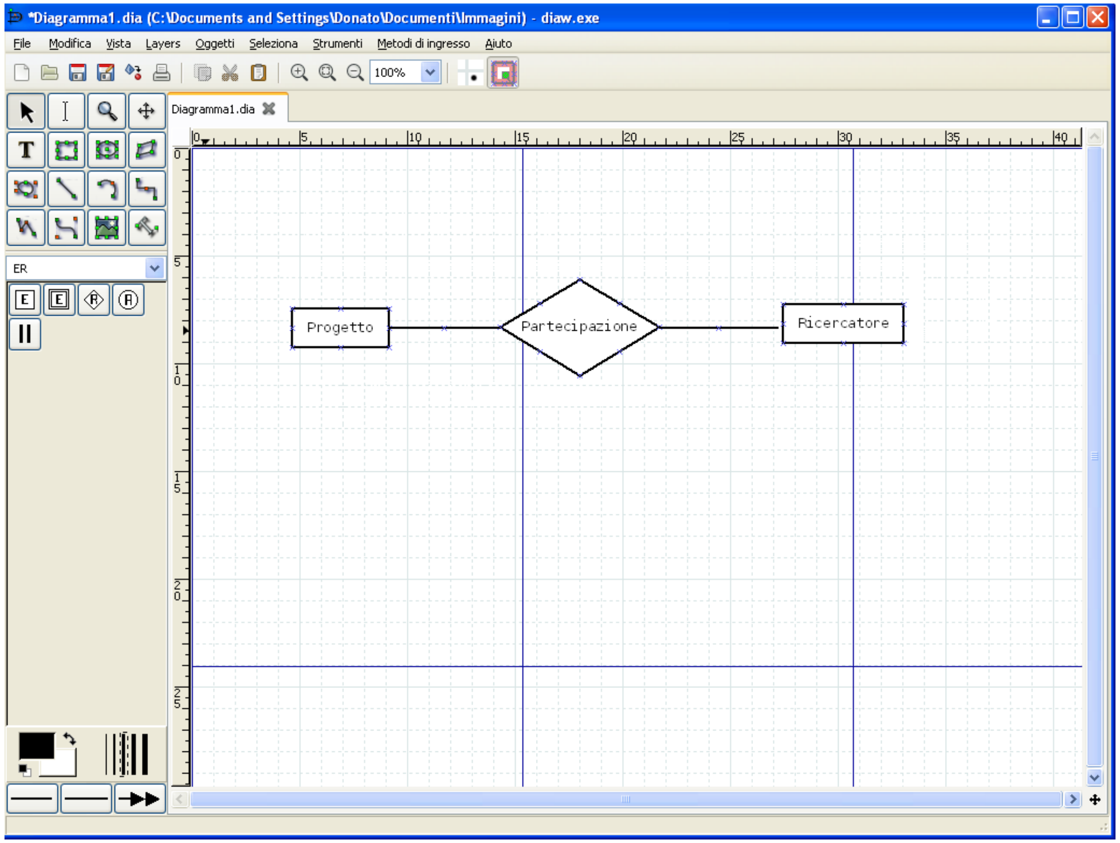


*Diagramma1.dia (C:\Documents and Settings\Donato\Documenti\Immagini) - diaw.exe
File Modifica Vista Layers Oggetti Seleziona Strumenti Metodi di ingresso Aiuto
100%
Diagramma1.dia
ER
Progetto
Partecipazione
Ricercatore

CASE tools for the design of Databases

The features offered by the various CASE vary greatly from one product to another, but there are some basic components that are present in all systems:

- A *graphic interface* with which directly manipulate the Entity-Relationship diagrams represented in diagrammatic form;
- A *data dictionary* centralized that stores information on various schema concepts (entities, attributes, relationships, integrity constraints, etc.).

CASE tools for the design of Databases

- A series of *integrated tools* running, automatically or through user interaction, specific tasks of planning
  - automatic layout of diagrams,
  - verification of correctness and completeness,
  - schema quality analysis,
  - automatic production of code for the implementation of the database.

CASE tools for the design of Databases

Many CASE tools can be integrated directly with the database management systems.

Other systems provide an activity of requirements analysis support.

Others provide libraries of predefined generic projects or previously developed that can be used as a starting point for a new project.

CASE tools for the design of Databases

Regarding the conceptual design, it is generally possible to use views strategies.

Many CASE allow to proceed in a top-down manner only partially defining certain schema concepts and then refine them later.

Other systems allow us to define and manipulate views separately, or portions of a basic pattern, automatically propagating in the base schema changes made on derivative schemes. This allows us to proceed bottom-up.

The UML model

UML

UML™ (*Unified Modeling Language*) is a visual language used to

- define
- design
- achieve
- document

communicate

systems (software) using an object oriented approach.

It is a language of representation of systems

It is universal: can represent heterogeneous systems architecture, technologies, application type (management, real-time, ...)

UML support

- the design of a new system
- the documentation of an existing system

UML

It can be used for the design of many different systems, from web to the more traditional systems

It can be used in many phases of the SW life cycle

- in the customer-supplier relationship
- engineer-engineer

without getting lost in the details of a programming language

Object-oriented modeling

In object-oriented modeling the building blocks are the objects and classes.

- An object is something usually comes from the vocabulary of the problem space and the solution space.
- A class is a description of a set of homogeneous objects.
- Every object has an identity, a state and a behavior.

Display, specify, construct, and document object-oriented systems is exactly the purpose of UML.

What is UML

It is not a "method" of software development

- It does not indicate any decomposition of the software development into subtasks.
- Neither it provides guidance on its use in a methodology.
- This allows the use of UML in different development processes such as cascade systems, incremental, spiral, etc ...
- Using UML should therefore be accompanied by a methodology that specifies how to use it.

It is not a "programming language"

It is not a proprietary language

It is not linked either to a specific programming language, or to a specific CASE tool

Notation, syntax and semantics are standard

Who has defined and promoted

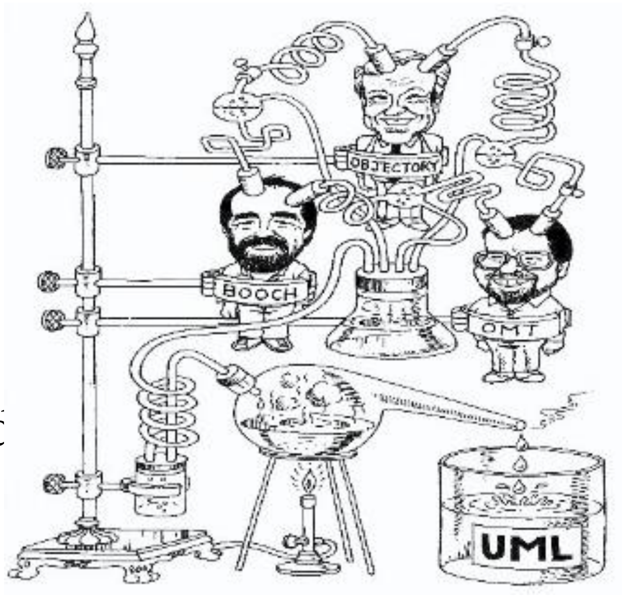
Standard OMG (Object Management Group), since November 1997

originators:

- Grady Booch
- Ivar Jacobson
- James Rumbaugh

References:

G.Booch, J. Rumbaugh, Jacobson I.
The Unified Modeling Language User guides. Ac



Who does it evolve

Its evolution is managed by the OMG and subject to well-defined procedures for each change

- first version: 1.1 (November 1997)
- Current version: 2.0 (2005)
- official documents: www.omg.org
- other UML resources: www.uml.org

UML Objectives

Provide the user with an expressive specification language, visual and quick to use

Offer extensibility mechanisms and language specialization

Being independent of programming languages

Producing a better SW through the use of modeling which allows us to:

- abstracting and simplifying the most complex projects
- view a system (how it is or how you would like)
- specify the structure and behavior of the system
- define guidelines for the construction of a system
- document the decisions taken

Data modeling in UML

UML is more than a set of graphic symbols. Each has a symbol in UML has a well-defined semantics that allows the unambiguous interpretation between various stakeholders and software tools.

UML is sometimes used as an alternative to the ER model for the conceptual representation of the data.

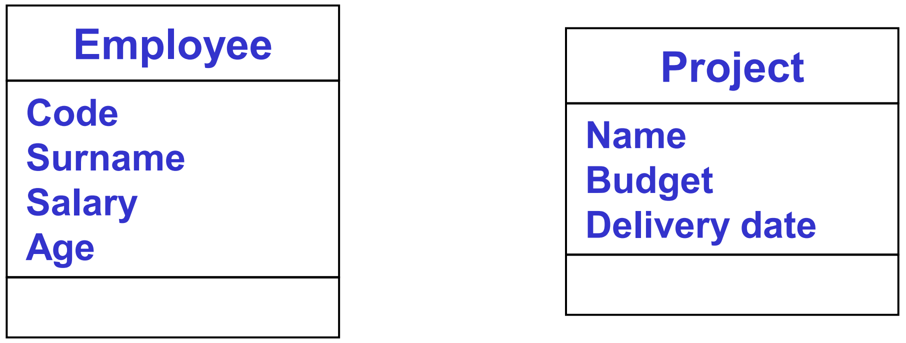
It makes use of class diagrams.

Change diagrammatic representation but not the approach to design.

Let's see how we can represent conceptual schemes with UML.

Classes

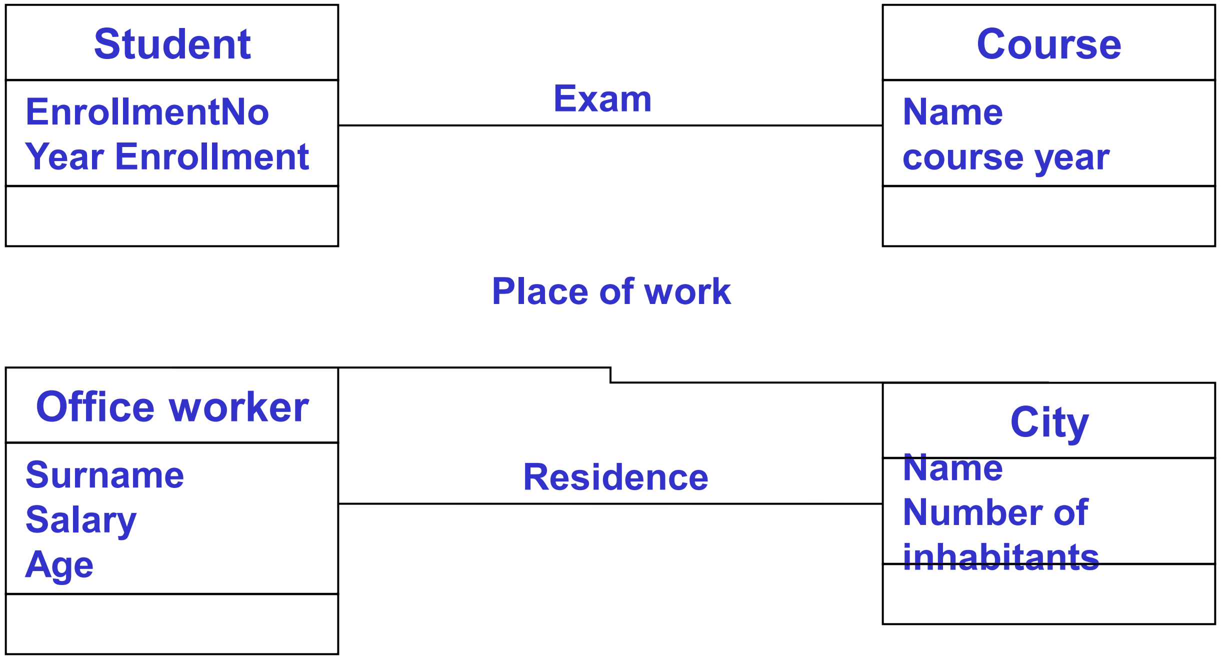
They are the main components of the class diagrams and substantially correspond to the ER model entities.



For a class, you can specify its methods. You can also specify the visibility of attributes and methods. These aspects are not considered in the data modeling.

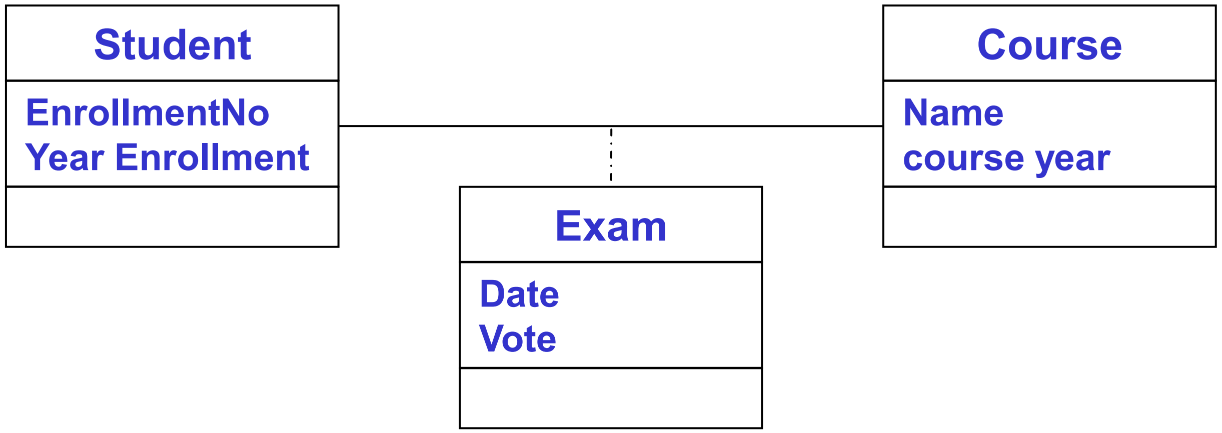
Associations

They correspond to the relationships in the ER model.



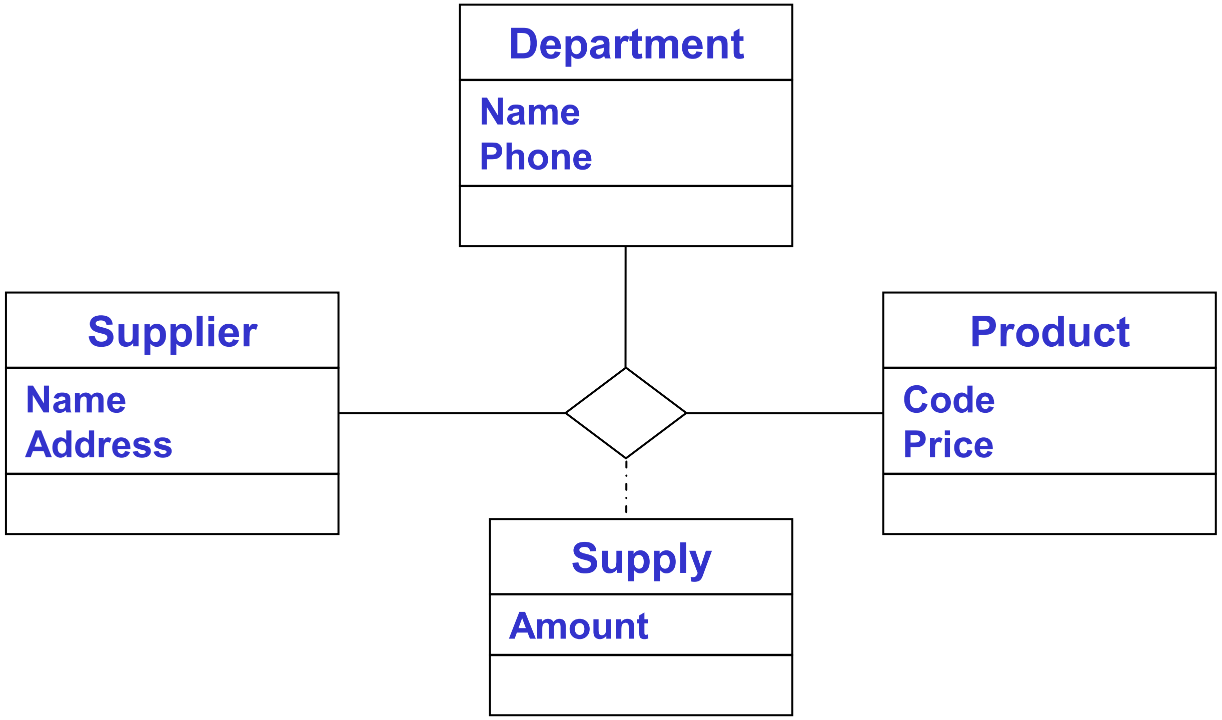
Association Class

You can not assign attributes to the associations. To do this, it is necessary to use the so-called association classes that describe properties of an association.
In this case it is not necessary to assign a name to the association.



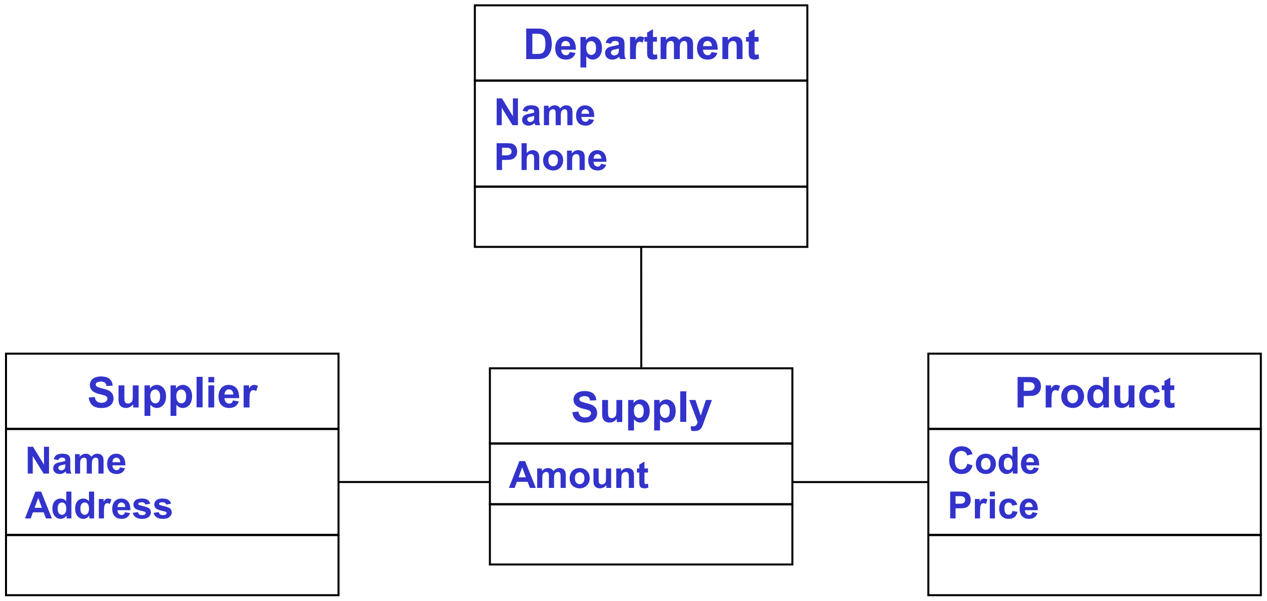
Association ternary

The symbol for the ternary association is similar to that of the ER model.



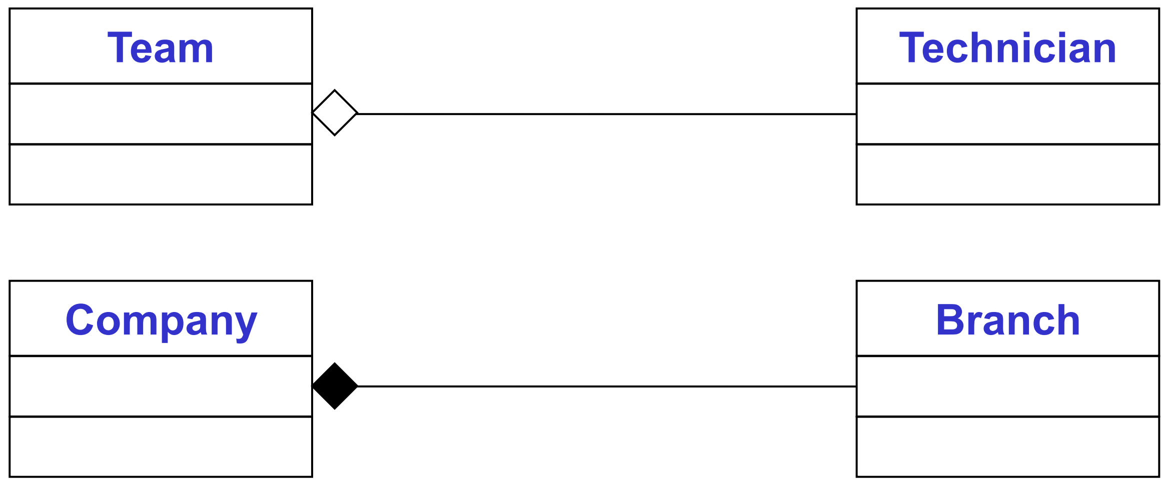
Reification of a relationship

The n-ary associations are used rarely in class diagrams. Usually they are reified.



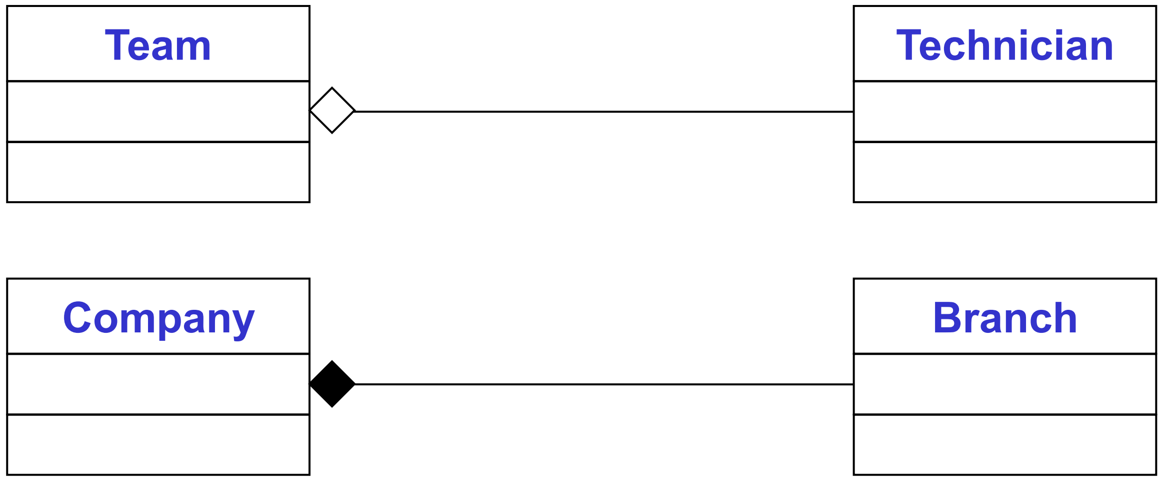
Aggregation and composition

In UML is a specific notation to specify associations that are aggregations of concepts, namely that they define a relationship between a composite concept and one or more concepts that constitute a part thereof.



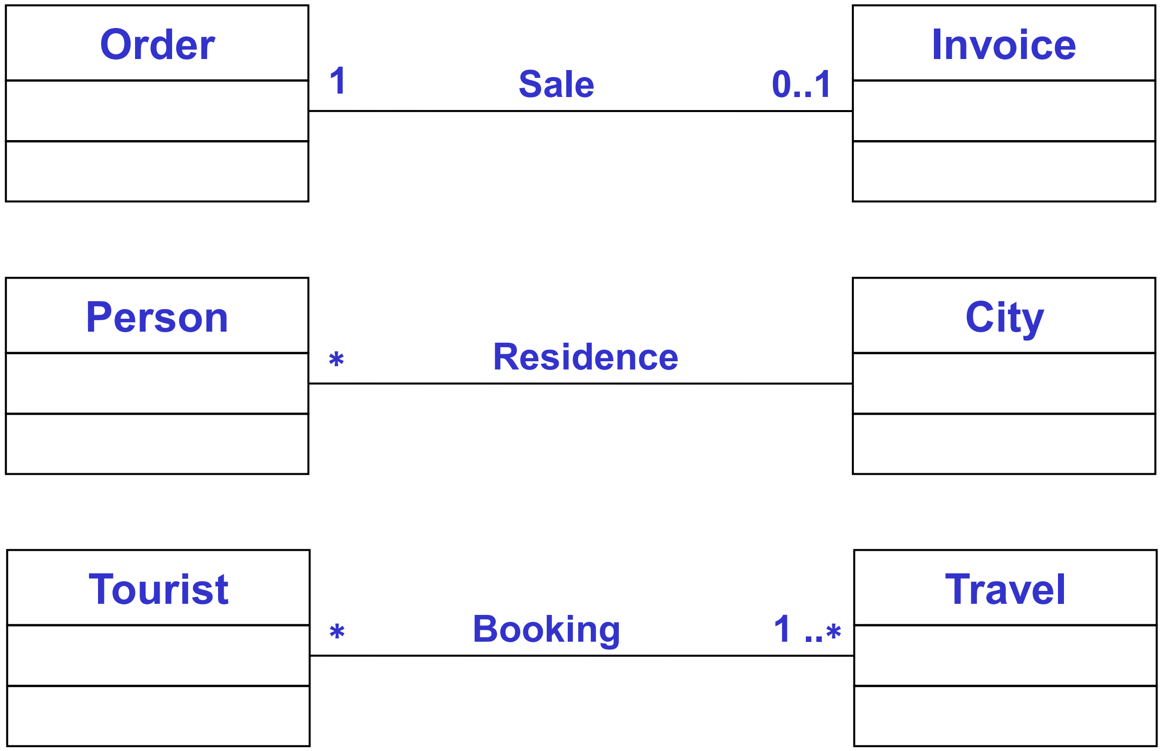
Aggregation and composition

The rhombus is blank if an object of class "part" can exist without having to belong to an object of class "aggregating", otherwise it is black and the combination is called composition.



Associations with multiplicity

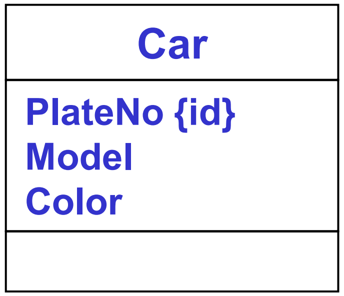
In UML it is possible to represent the cardinality or multiplicity of an association with the same model of the ER mode. They change, however, the conventions.

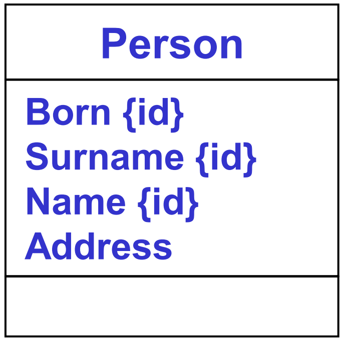


Identifiers

In UML there is no notation to express identifiers of classes, since the instances of a class, called objects, have their own identifier (OID). To represent (in non-standard way) an identifier a construct called user constraint is used. They are user-defined (non-default).

The attributes of an identifier are indicated by {Id}. You can specify a single identifier for class.

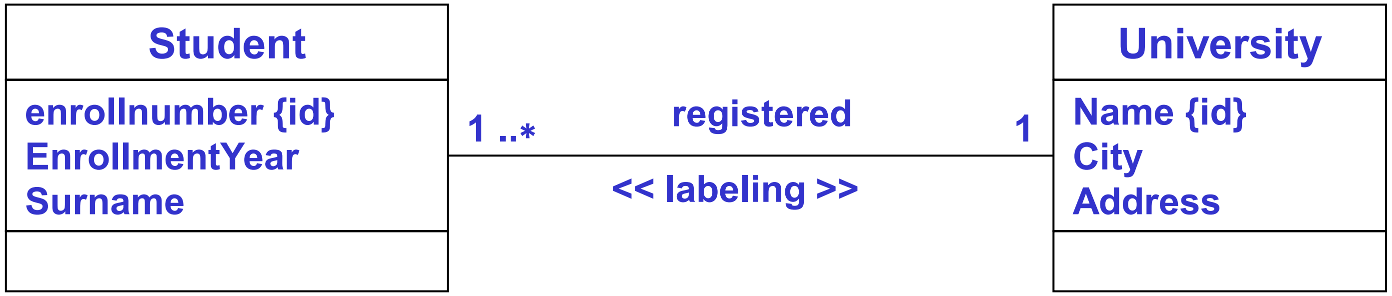




External ID

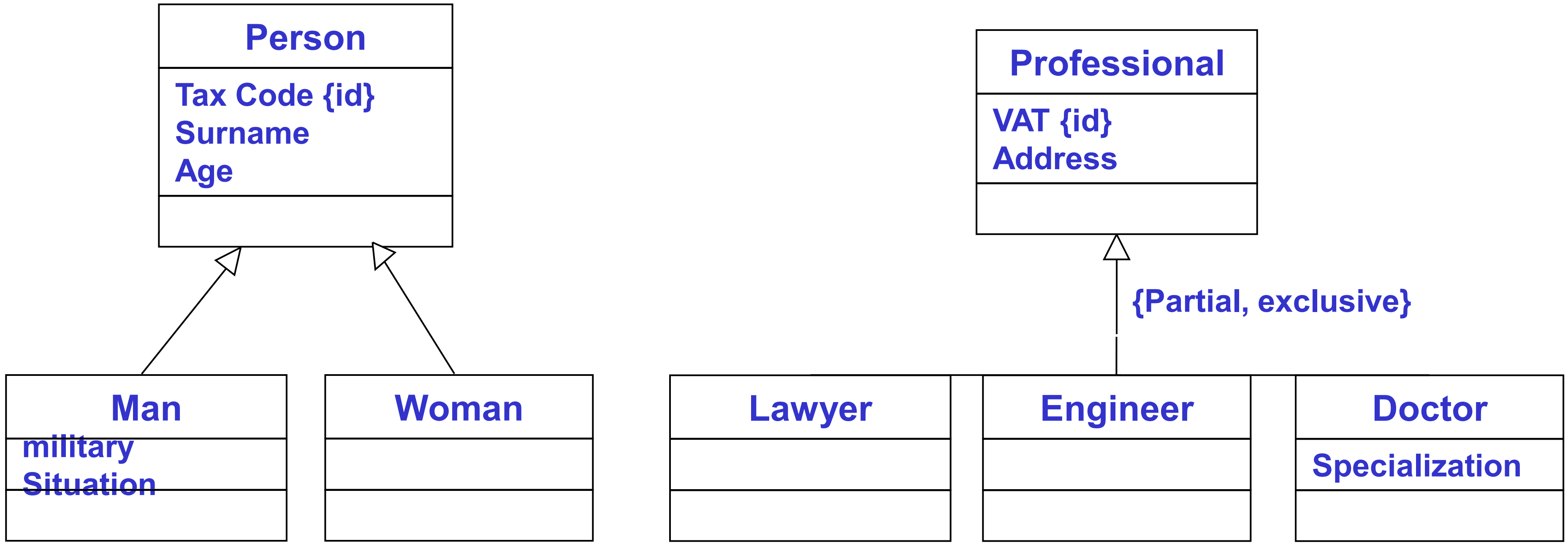
To represent an external identifier a stereotype should be used.

The attributes of an identifier are indicated by {Id}. You can specify a single identifier for class.

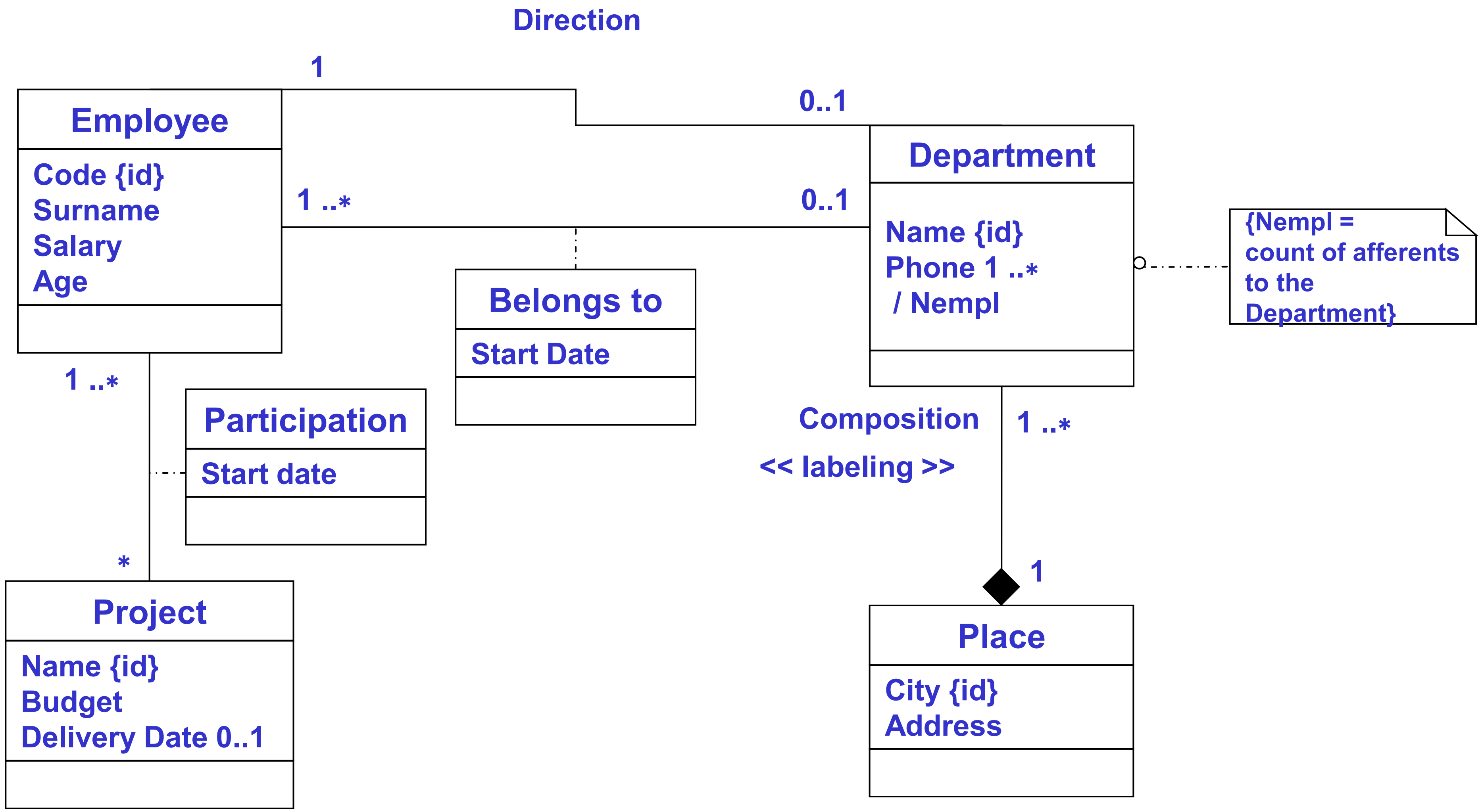


Generalizations

Generalizations can be represented as in the ER model. Any property (totality, exclusivity, etc.) may be represented by user constraints listed to the left of the generalizations.



A conceptual schema in UML



The derived attributes are indicated with / and a note is used to describe the way to obtain it.

Exercise

A public relations agency organizes banquets for large events or for small events (congresses, weddings, affirmations, etc.) on behalf of clients, who may be companies or individuals. For the customer we know the fiscal code, possible VAT registration, and personal data.
For each banquet we know the number of guests, the list of guests, the date, the menu, the cost, and the restaurant at which it is held. Among the restaurants you can see the characteristic ones, which can offer specialties. The choice of the restaurant should also be based on the number of places available at the restaurant. Among the menus there are those containing, among the various courses, at least one specialty offered by the characteristic restaurants. The choice of the menu will also be based on the cost.

Solution: Requirements Analysis

Analysis of specific requirements:

· choose the right level of abstraction

- for *personal data* of customer we intends name / company, address, phone number
- for *guest list* it means the list of surnames and names of the invited

· linearize phrases and break down those articulated:

The phrase:

"The public relations agency organizes banquets for large events or for small events (congresses, weddings, affirmations, etc.) on behalf of clients, who may be companies or individuals."

It appears to be excessively complex. Additionally, in the same sentence both the agency and the customers are involved.

Solution: Requirements Analysis

It therefore prefers to break it down into the following two sentences:

"The public relations agency organizes banquets for large events or for small events (conferences, weddings, confirmations, etc.)."

and

"Customers of the agency can be companies or individuals."

- standardize sentences → no change
- identify homonyms and synonyms → no change
- making explicit references between terms:

the terms 'cost' in line 4 and line 9, refers to two different entities so it is better to be more explicit:

total cost of the banquet (line 4)

cost of the menu (line 9)

Solution: Requirements Analysis

- build a glossary of terms

TERM	DESCRIPTION	CONNECTION
banquet	Friendly event organized by the PR agency	Customer, restaurant, menu
customer	A PR agency customer who can order a banquet	banquet
restaurant	Localpublicat which was held on thebanquet	banquet menu
Menu	list of the courses offered by the restaurant and / or offered to the banquet	restaurant, banquet

Solution: Requirements Analysis

- reorganize sentences for specific concepts (you can replicate)

- General phrases:

The public relations agency, organizes banquets for large events or for small events (conferences, weddings, confirmations, etc.).

- Phrases related to customers:

Customers of the agency can be individuals or companies. Of the customer we know the tax code and / or VAT number, name / company, address, and telephone number.

- Phrases related to banquets:

For each banquet we know the number of guests, the list of names and surnames of the guests, the date, the menu, the total cost of the banquet and the restaurant where it is held

- Phrases related to restaurants:

Among the restaurants we distinguish those characteristic that may offer particular specialties. The choice of restaurant should also be based on the number of places available at the restaurant.

- Phrases related to menus:

The menus have a list of courses. Among the menus we distinguish those containing, among other courses, at least one specialty offered by characteristic restaurants. The choice of the menu will be based on its cost

Solution: Conceptual Design

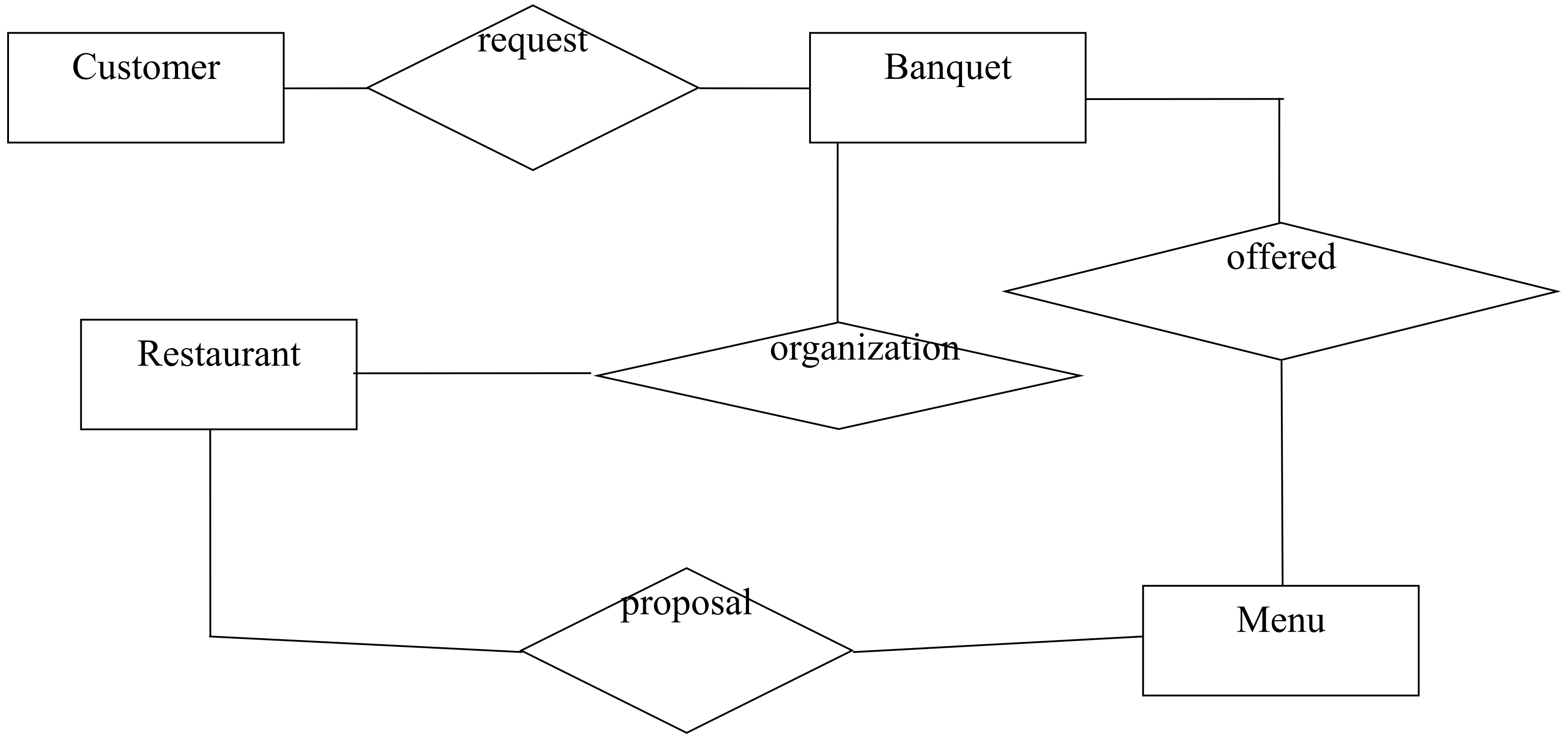
- represent the specific ER with a scheme

The strategy followed in the phase of conceptual modeling is the HYBRID one: starting from the specifications we will represent all the information in an initial skeleton using a few abstract concepts. Subsequently, each entity of such a scheme has to be refined and the different patterns obtained will be integrated, leading to the final ER scheme, much more detailed than the initial one.

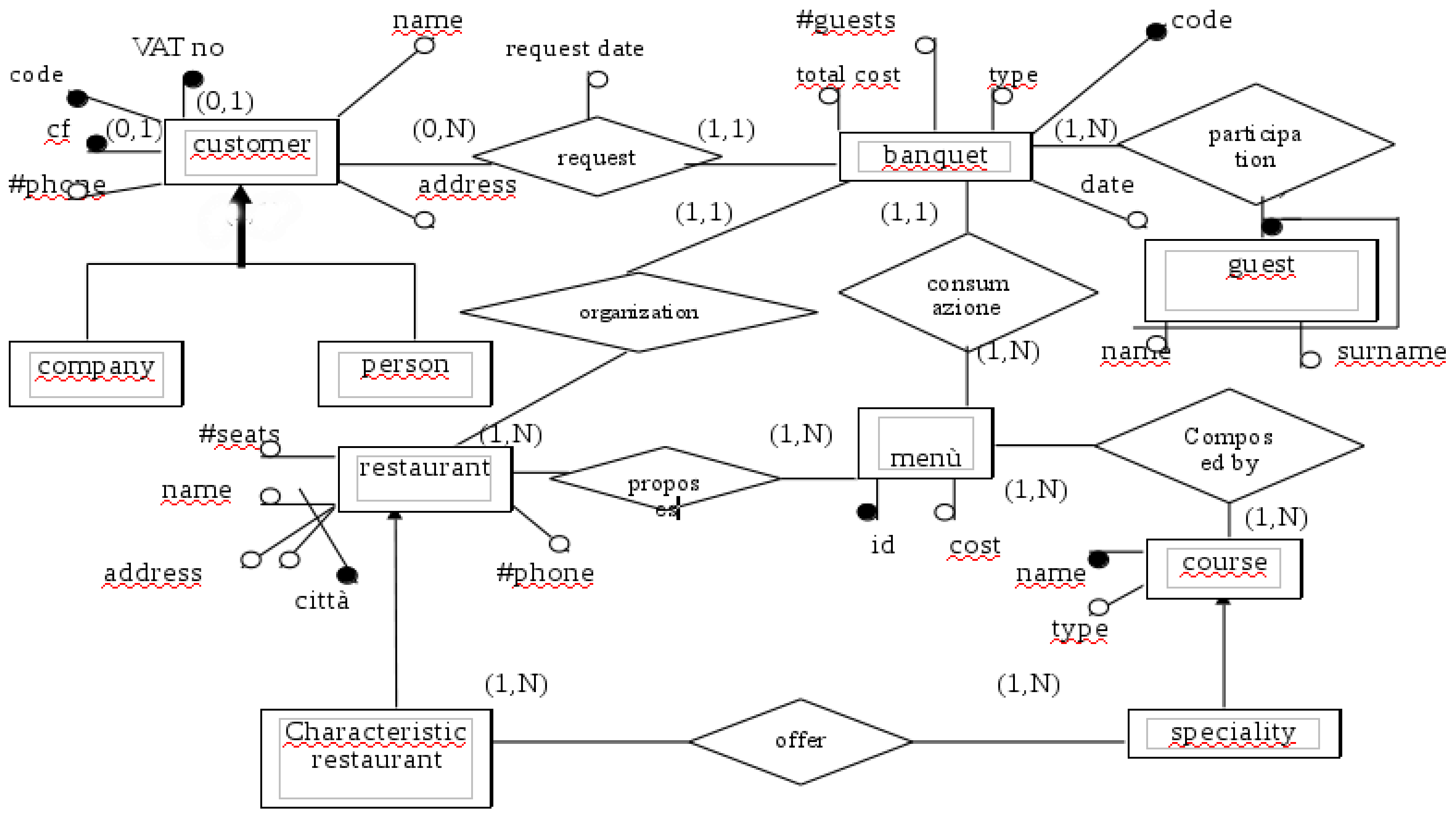
Solution: Conceptual Design

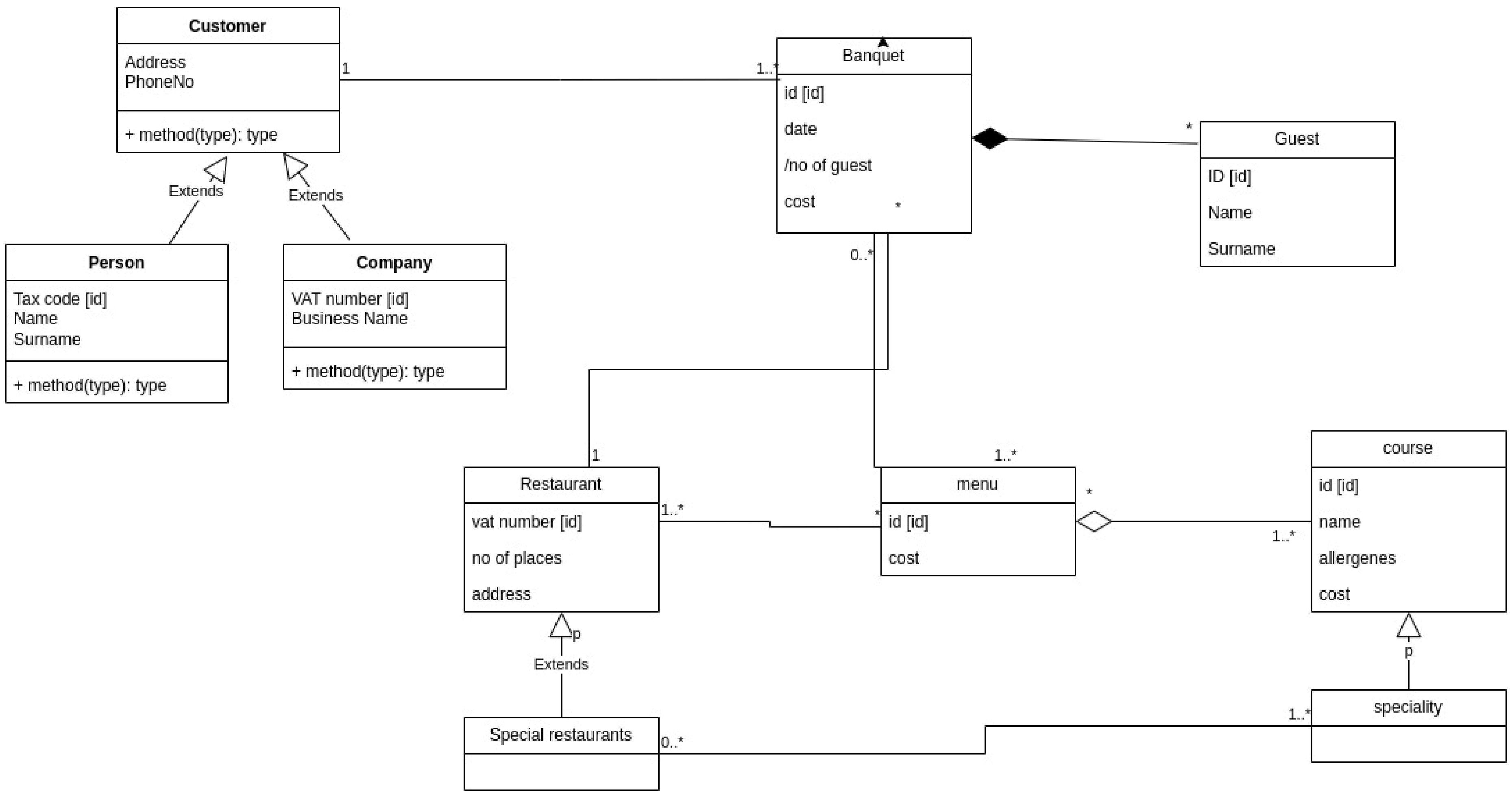
- represent with a specific ER diagram (cont).

ER FIRST SCHEDULE



..after several steps




Customer
Address
PhoneNo
+ method(type): type
Extends
Extends
Person
Tax code [id]
Name
Surname
+ method(type): type
Company
VAT number [id]
Business Name
+ method(type): type
1
1..*
Banquet
id [id]
date
/no of guest
cost
*
*
Guest
ID [id]
Name
Surname
0..*
1
Restaurant
vat number [id]
no of places
address
1..*
*
1..*
menu
id [id]
cost
*
1..*
course
id [id]
name
allergenes
cost
p
Extends
p
Special restaurants
0..*
1..*
speciality

Solution: Conceptual Design

Choices made in the conceptual design phase

- The database was completed with reasonable attributes for each entity.
- For the specializations "company" and "person" entity "customer" (they constitute a total and exclusive generalization hierarchy) it was not considered appropriate to include the attributes, since the operations performed on the database refers to "customer", without making distinctions nor shall special news about the "companies" or the "people".

Solution: Conceptual Design

Choices made in the conceptual design phase

- The "guests" are represented as weak entity dependent on "banquet": this is due to the fact that the entity "guest" existentially dependent on the presence of the entity instance "banquet". Also the guests of a banquet would hardly be known to the agency with data that can be an identifier: it is therefore essential to add the name and surname of the person invited to the banquet which the call relates, in order to uniquely identify the invited.
- The characteristic restaurants and specialties were represented by subsets.

Solution: Conceptual Design

The scheme will be completed with the relevant documentation to the constraints not expressed in the ER diagram.

Example:

- The number of guests at a banquet is less than or equal to the number of seats available in the restaurant where the banquet is organized
- Menus for a banquet should be between those offered by the restaurant at which the banquet is given.

Exercise

Define a ER diagram for a library, with the following specifications:

Subject of loans are exemplary object (also called copies) of individual volumes, identified through an inventory number; each volume is related to a specific edition (which can be divided into multiple volumes, also in a different way from the other editions) of a work a volume can be present in multiple copies an edition is characterized by the work, the series and the year summarizing and exemplifying, you can borrow the second copy of the third volume of "Les Miserables", Mondadori edition, Oscar series, 1975 each series has a name and a code and publisher each publisher has a name and a code each work has a title, an author and year of first publication for each current loan (those ended are of no interest), the date of return and the user they are relevant (the user can have multiple volumes simultaneously borrowed), with identification number, name and telephone number

References

P. Atzeni, S. Ceri, S. & R. Paraboschi Torlone
Databases: models and query languages. Third edition.
McGraw-Hill Books Italy, 2009.
Chapter 7